

Ts. **Nguyễn Đình Thúc** (chủ biên)





TRÍ TUỆ NHÂN TẠO

Lập trình tiến hóa

Cấu trúc dữ liệu + Thuật giải di truyền = Chương trình tiến hóa

- **Thuật giải di truyền**
- **Tối ưu số**
- **Tối ưu tổ hợp**



NHÀ XUẤT BẢN GIÁO DỤC

5/289-00 PIK08n0

MỤC LỤC

Mở đầu

Cấu trúc dữ liệu + Thuật giải di truyền = Chương trình tiến hoá

Thuật ngữ *Chương trình tiến hoá* trong công thức trên là khái niệm dùng để chỉ các chương trình máy tính có sử dụng thuật toán tìm kiếm và tối ưu hóa dựa trên nguyên lý *tiến hóa tự nhiên*. Ta gọi chung các thuật toán như thế là *thuật toán tiến hóa*. Dưới đây là một số thuật toán tiến hóa đã được công bố.

- Qui hoạch tiến hóa – EP, do D.B. Pogel đề xuất. Có thể diễn tả EP đơn giản như sau: Cho một lớp các phương pháp khả dĩ giải quyết được một (số) phần của vấn đề. Dựa vào qui luật tiến hóa, tìm một phương pháp liên hợp đủ khả năng giải quyết trọn vẹn vấn đề đó.
- Chiến lược tiến hóa, do T. Baeck, F.H. Hofmeister và H.P. Schwefel đề xuất. Thuật toán này dựa trên một số chiến lược ban đầu, tiến hóa để tạo ra những chiến lược mới phù hợp với môi trường thực tế một cách tốt nhất.
- Thuật giải di truyền do D.E. Goldberg đề xuất, được L. Davis và Z. Michalevicz phát triển. Đây là thuật toán tiến hóa chính chúng tôi đề cập trong cuốn sách này.

Thuật giải di truyền, cũng như các thuật toán tiến hóa nói chung, hình thành dựa trên quan niệm cho rằng, quá trình tiến hóa tự nhiên là quá trình hoàn hảo nhất, hợp lý nhất, và tự nó đã mang tính tối ưu. Quan niệm này có thể được xem như một tiên đề đúng, không chứng minh được, nhưng phù hợp với thực tế khách quan. Quá trình tiến hóa thể hiện tính tối ưu ở chỗ, thế hệ sau bao giờ cũng tốt

hơn (phát triển hơn, hoàn thiện hơn) thế hệ trước. Tiến hóa tự nhiên được duy trì nhờ hai quá trình cơ bản: sinh sản và chọn lọc tự nhiên. Xuyên suốt quá trình tiến hóa tự nhiên, các thế hệ mới luôn được sinh ra để bổ sung thay thế thế hệ cũ. Cá thể nào phát triển hơn, thích ứng hơn với môi trường sẽ tồn tại. Cá thể nào không thích ứng được với môi trường sẽ bị đào thải. Sự thay đổi môi trường là động lực thúc đẩy quá trình tiến hóa. Ngược lại, tiến hóa cũng tác động trở lại góp phần làm thay đổi môi trường.

Các cá thể mới sinh ra trong quá trình tiến hóa nhờ sự lai ghép ở thế hệ cha-mẹ. Một cá thể mới có thể mang những tính trạng của cha-mẹ (*di truyền*), cũng có thể mang những tính trạng hoàn toàn mới (*đột biến*). Di truyền và đột biến là hai cơ chế có vai trò quan trọng như nhau trong tiến trình tiến hóa, dù rằng đột biến xảy ra với xác suất nhỏ hơn nhiều so với hiện tượng di truyền. Các thuật toán tiến hóa, tuy có những điểm khác biệt, nhưng đều mô phỏng bốn quá trình cơ bản: *lai ghép, đột biến, sinh sản* và *chọn lọc tự nhiên.*

Quá trình lai ghép (phép lai)

Phép lai là quá trình hình thành nhiễm sắc thể mới trên cơ sở các nhiễm sắc thể cha-mẹ, bằng cách ghép một hay nhiều đoạn gen của hai (hay nhiều) nhiễm sắc thể cha-mẹ với nhau. Phép lai xảy ra với xác suất p_c, có thể mô phỏng như sau:

- Chọn ngẫu nhiên hai (hay nhiều) cá thể bất kỳ trong quần thể. Giả sử các nhiễm sắc thể của cha-mẹ đều có m gen.
- Tạo một số ngẫu nhiên trong khoảng từ *1* đến $m-1$ (ta gọi là điểm lai). Điểm lai chia các chuỗi cha-mẹ dài m thành hai nhóm chuỗi con dài m_1 và m_2. Hai chuỗi nhiễm sắc thể con mới sẽ là $m_{11}+m_{22}$ và $m_{21}+m_{12}$.

- Đưa hai cá thể mới này vào quần thể để tham gia các quá trình tiến hóa tiếp theo.

Quá trình đột biến (phép đột biến)

Đột biến là hiện tượng cá thể con mang một (số) tính trạng không có trong mã di truyền của cha-mẹ. Phép đột biến xảy ra với xác suất p_m, nhỏ hơn rất nhiều so với xác suất lai p_c. Phép đột biến có thể mô phỏng như sau:

- Chọn ngẫu nhiên một cá thể bất kỳ cha-mẹ trong quần thể.
- Tạo một số ngẫu nhiên k trong khoảng từ *1* đến m, $1 \le k \le m$.
- Thay đổi gen thứ k và trả cá thể này về quần thể để tham gia quá trình tiến hóa tiếp theo.

Quá trình sinh sản và chọn lọc (phép tái sinh và phép chọn)

Phép tái sinh là quá trình trong đó các cá thể được sao chép trên cơ sở độ thích nghi của nó. Độ thích nghi là một hàm gán một giá trị thực cho các cá thể trong quần thể. Quá trình này có thể được mô phỏng như sau:

- Tính độ thích nghi của từng cá thể trong quần thể hiện hành, lập bảng cộng dồn các giá trị thích nghi (theo số thứ tự gán cho từng cá thể). Giả sử quần thể có n cá thể. Gọi độ thích nghi của cá thể thứ i là F_i, tổng dồn thứ i là F_{ti}, tổng độ thích nghi của toàn quần thể là F_m.
- Tạo một số ngẫu nhiên F trong đoạn từ *0* đến F_m.
- Chọn cá thể thứ k đầu tiên thỏa $F \ge F_{tk}$ đưa vào quần thể của thế hệ mới.

Phép chọn là quá trình loại bỏ các cá thể xấu trong quần thể để chỉ giữ lại trong quần thể các cá thể tốt. Phép chọn có thể được mô phỏng như sau:

- Sắp xếp quần thể theo thứ tự độ thích nghi giảm dần.
- Loại bỏ các cá thể cuối dãy để chỉ giữ lại *n* cá thể tốt nhất. Ở đây, ta giả sử quần thể có kích thước cố định *n*.

Một thuật giải di truyền, giải một bài toán được cho phải có năm thành phần sau:

- Một cấu trúc dữ liệu I biểu diễn không gian lời giải của bài toán,
- Phương pháp khởi tạo quần thể ban đầu **P(0)**,
- Hàm định nghĩa độ thích nghi **eval(.)** đóng vai trò môi trường,
- Các phép toán di truyền như đã mô phỏng trên,
- Và các tham số thuật giải di truyền sử dụng (kích thước quần thể, xác suất lai, đột biến,...)

Hình 0.1. trình bày một cấu trúc thuật giải di truyền tổng quát.

Thuật giải di truyền

Bắt đầu

t = 0;

Khởi tạo P(t);

Tính độ thích nghi cho các cá thể thuộc P(t);

Khi (điều kiện dừng chưa thoả) **lặp**

t = t + 1;

Tái sinh P'(t) từ P(t)

Lai Q(t) từ P(t-1);

Đột biến R(t) từ P(t-1);

Chọn lọc P(t) từ P(t-1) ∪ Q(t) ∪ R(t) ∪ P(t);

Hết lặp

Kết thúc

Hình 0.1. Một thuật giải di truyền

Trong lập trình tiến hóa, khi giải một bài toán đặt ra, cần tận dụng tối đa tri thức về bài toán đó để chương trình tiến hóa đạt được hiệu quả cao nhất có thể. Việc tận dụng tri thức bài toán có thể được thể hiện (1) qua việc xây dựng một cấu trúc dữ liệu hợp lý sao cho việc xây dựng các phép toán di truyền được tự nhiên và hiệu quả nhất (2) hay qua việc sử dụng phương pháp đã và đang được sử dụng để giải bài toán này và kết hợp chúng với thuật giải di truyền (3) và cách tận dụng hay nhất là tận dụng cả 2 cách trên trong 1 chương trình tiến hóa; đây là cách được nhiều nhà nghiên cứu ứng dụng lập trình tiến hóa sử dụng nhất. Tuy nhiên, để không bị phân tán, chúng tôi chỉ tập trung vào cách thứ nhất: tận dụng tri thức để biểu diễn cấu trúc dữ liệu và xây dựng các phép di truyền. Hy vọng, khi

đã nắm vững các kỹ thuật cơ bản của lập trình tiến hóa được trình bày trong cuốn sách này, bạn có thể tận dụng tri thức bài toán theo cách (2) và (3) để giải bài toán của bạn.

Nội dung của cuốn sách được xây dựng trên cơ sở hai tài liệu chính: cuốn Evolutionary Algorithms in Theory and Practice của Thomas Back (1996) và cuốn Genetic Algorithms + Data Structures = Evolution Programs của Zbigniew Michalewicz (1999) và được chia làm ba phần chính và hai phụ lục:

- **Phần 1: Thuật giải di truyền.** Phần này trình bày chi tiết về thuật giải di truyền cũng như nguyên lý và cơ chế hoạt động của nó.
- **Phần 2: Tối ưu số.** Phần này trình bày cách áp dụng thuật giải di truyền giải các bài toán tối ưu số. Chúng tôi cũng trình bày một số phương pháp giải quyết các ràng buộc của chương trình tiến hóa trong một bài toán qui hoạch phi tuyến tổng quát.
- **Phần 3: Tối ưu tổ hợp.** Các bài toán tổ hợp thuộc lớp bài toán NP-đủ được xem xét và giải quyết bằng thuật giải di truyền sẽ được trình bày và phân tích trong phần này.

Cuối cùng, hai phụ lục sẽ trình bày các khía cạnh lý thuyết cũng như những khía cạnh ứng dụng liên quan đến thuật giải di truyền gốc và các thuật giải di truyền cải tiến. Ở đây, chúng tôi cũng trình bày tóm tắt một số thuật toán tiến hóa khác.

Phần 1

THUẬT GIẢI DI TRUYỀN

Chương 1

THUẬT GIẢI DI TRUYỀN : CÁC KHÁI NIỆM CƠ BẢN

Với khả năng hiện nay, máy tính đã giúp giải được rất nhiều bài toán khó mà trước kia thường bó tay. Mặc dù vậy, vẫn còn một số lớn các bài toán rất thú vị nhưng chưa có thuật giải hợp lý để giải chúng. Trong số đó, các bài toán tối ưu là những bài toán thường xuyên gặp phải trong các ứng dụng thực tiễn.

Trong thực tiễn, có nhiều bài toán tối ưu quan trọng đòi hỏi những thuật giải chất lượng cao. Ví dụ, ta có thể áp dụng phương pháp *mô phỏng luyện thép* để giải bài toán tìm đường đi ngắn nhất cho xe cứu hoả hay bài toán người du lịch.... Cũng có nhiều bài toán tối ưu tổ hợp (trong đó có nhiều bài đã được chứng minh là thuộc loại *NP - đủ)* có thể được giải gần đúng trên máy tính hiện đại bằng kỹ thuật Monte-Carlo.

Nói chung, bài toán tối ưu có thể được xem như bài toán tìm kiếm giải pháp (tốt nhất) trong không gian (vô cùng lớn) các giải pháp. Khi không gian tìm kiếm nhỏ, các phương pháp cổ điển như trên cũng đủ thích hợp; nhưng khi không gian lớn cần phải dùng đến những kỹ thuật *Trí Tuệ Nhân Tạo* đặc biệt. *Thuật giải Di Truyền (GA)* là một trong những kỹ thuật đó. GA là một loại thuật giải mô phỏng các hiện tượng tự nhiên: *kế thừa* và *đấu tranh sinh tồn* để cải tiến lời giải và khảo sát không gian lời giải. Khái niệm kế thừa và đấu tranh sinh tồn được giải thích qua thí dụ về sự tiến hóa của một quần thể thỏ như sau:

Có một quần thể thỏ. Trong số đó có một số con nhanh nhẹn và thông minh hơn những con khác. Những chú thỏ nhanh nhẹn và

thông minh có xác suất bị chồn cáo ăn thịt nhỏ hơn, do đó chúng tồn tại để làm những gì tốt nhất có thể: Tạo thêm nhiều thỏ tốt. Dĩ nhiên, một số thỏ chậm chạp đần độn cũng sống chỉ vì may mắn. Quần thể những chú thỏ còn sống sót sẽ bắt đầu sinh sản. Việc sinh sản này sẽ tạo ra một hỗn hợp tốt về "nguyên liệu di truyền thỏ": Một số thỏ chậm chạp có con với những con thỏ nhanh, một số thỏ nhanh với thỏ nhanh, một số thỏ thông minh với thỏ đần độn, vv... Và trên tất cả, thiên nhiên thỉnh thoảng lại ném vào một con thỏ 'hoang dã' bằng cách làm đột biến nguyên liệu di truyền thỏ. Những chú thỏ con, do kết quả này, sẽ nhanh hơn và thông minh hơn những con trong quần thể gốc vì có nhiều bố mẹ nhanh nhẹn và thông minh hơn đã thoát chết khỏi chồn cáo. (Thật hay là những con chồn cũng trải qua những tiến trình tương tự - nếu không những con thỏ sẽ trở nên nhanh và thông minh đến nỗi những con chồn không thể bắt chúng được).

Khi tìm kiếm lời giải tối ưu, thuật giải di truyền cũng thực hiện các bước tương ứng với câu chuyện đấu tranh sinh tồn của loài thỏ.

Thuật giải di truyền sử dụng các thuật ngữ vay mượn của di truyền học. Ta có thể nói về các cá thể (hay *kiểu gen*, *cấu trúc*), trong một quần thể; những cá thể này cũng còn được gọi là các *chuỗi* hay các *nhiễm sắc thể*. Điều này có thể gây chút lẫn lộn: mỗi tế bào của một cơ thể của một chủng loại đã cho, mang một số những nhiễm sắc thể nào đó (thí dụ, người có 46 nhiễm sắc thể) nhưng trong thuật giải di truyền, ta chỉ nói về những cá thể có một nhiễm sắc thể. Các nhiễm sắc thể được tạo thành từ các đơn vị - *các gen* – biểu diễn trong một chuỗi tuyến tính; mỗi gen kiểm soát một (số) đặc trưng. Gen với những đặc trưng nhất định có vị trí nhất định trong nhiễm sắc thể. Bất cứ đặc trưng nào của mỗi cá thể có thể tự biểu hiện một cách phân biệt; và gen có thể nhận một số giá trị khác nhau (các giá trị về tính năng).

Mỗi kiểu (nhóm) gen (ta gọi là một nhiễm sắc thể) sẽ biểu diễn một lời giải của bài toán đang giải (ý nghĩa của một nhiễm sắc thể cụ thể được người sử dụng xác định trước); một tiến trình tiến hóa được thực hiện trên một quần thể các nhiễm sắc thể tương ứng với một quá trình tìm kiếm lời giải trong không gian lời giải. Tìm kiếm đó cần cân đối hai mục tiêu (có vẻ mâu thuẫn nhau): Khai thác những lời giải tốt nhất và khảo sát không gian tìm kiếm. *Leo đồi* là một thí dụ về chiến lược cho phép khai thác và cải thiện lời giải tốt nhất hiện hành; nhưng, leo đồi lại bỏ qua việc khảo sát không gian tìm kiếm. Ngược lại, *tìm kiếm ngẫu nhiên* là một thí dụ điển hình của chiến lược khảo sát không gian tìm kiếm mà không chú ý đến việc khai thác những vùng đầy hứa hẹn của không gian. *Thuật giải di truyền (GA)* là phương pháp tìm kiếm (độc lập miền) tạo được sự cân đối đáng kể giữa việc khai thác và khảo sát không gian tìm kiếm.

Thực ra, GA thuộc lớp các thuật giải xác suất, nhưng lại rất khác những thuật giải ngẫu nhiên vì chúng kết hợp các phần tử tìm kiếm trực tiếp và ngẫu nhiên. Khác biệt quan trọng giữa tìm kiếm của GA và các phương pháp tìm kiếm khác là GA duy trì và xử lý một tập các lời giải (ta gọi là một quần thể) - tất cả những phương pháp khác chỉ xử lý một điểm trong không gian tìm kiếm. Chính vì thế, GA mạnh hơn các phương pháp tìm kiếm hiện có rất nhiều.

Đơn cử, ta so sánh GA với hai phương pháp tìm kiếm hiện được sử dụng rộng rãi: Leo đồi và Mô phỏng luyện thép.

Phương pháp leo đồi dùng kỹ thuật lặp và áp dụng cho một điểm duy nhất (điểm hiện hành trong không gian tìm kiếm). Trong mỗi bước lặp, một điểm mới được chọn từ lân cận của điểm hiện hành (vì thế leo đồi còn được gọi là phương pháp tìm kiếm lân cận hay tìm kiếm cục bộ). Nếu điểm mới cho giá trị (của hàm mục tiêu) tốt hơn, điểm mới sẽ trở thành điểm hiện hành. Nếu không, một lân

cận khác sẽ được chọn và thử. Quá trình trên sẽ dừng nếu không cải thiện thêm được cho lời giải hiện hành.

Rõ ràng là phương pháp leo đồi chỉ cung cấp các giá trị tối ưu cục bộ và những giá trị này phụ thuộc rất nhiều vào điểm khởi đầu. Hơn nữa, không có thông tin sẵn có về sai số tương đối (thỏa tối ưu toàn cục) của lời giải tìm được.

Để tăng cơ hội thành công, phương pháp leo đồi thường được thực hiện nhiều lần; mỗi lần với một điểm khởi đầu khác nhau (những điểm này không cần chọn ngẫu nhiên - một tập hợp các điểm khởi đầu của một lần thực thi phụ thuộc vào kết quả của những lần chạy trước đó).

Kỹ thuật mô phỏng luyện thép là một kỹ thuật khắc phục những bất lợi của phương pháp leo đồi: Lời giải không còn tùy thuộc nhiều vào điểm khởi đầu nữa và (thường là) gần với điểm tối ưu. Đạt được điều này là nhờ đưa vào xác suất nhận p. Xác suất p là hàm theo giá trị của hàm mục tiêu đối với điểm hiện hành và điểm mới, và một tham số điều khiển bổ sung, tham số "nhiệt độ" T. Nói chung, nhiệt độ T càng thấp thì cơ hội nhận điểm mới càng nhỏ. Khi thực hiện thuật giải, nhiệt độ T của hệ thống sẽ được hạ thấp dần theo từng bước. Thuật giải dừng khi T nhỏ hơn một ngưỡng cho trước; với ngưỡng này thì gần như không còn thay đổi nào được chấp nhận nữa.

Như đã đề cập, GA thực hiện tiến trình tìm kiếm lời giải tối ưu theo nhiều hướng, bằng cách duy trì một *quần thể* các lời giải, và thúc đẩy sự thành hình và trao đổi thông tin giữa các hướng này. Quần thể trải qua tiến trình trình tiến hóa: ở mỗi *thế hệ* lại *tái sinh* các lời giải tương đối "tốt", trong khi các lời giải tương đối "xấu" thì chết đi. Để phân biệt các lời giải khác nhau, hàm mục tiêu được dùng để đóng vai trò môi trường.

Cấu trúc của một thuật giải di truyền đơn giản tương tự với cấu trúc của bất kỳ chương trình tiến hóa nào (xem hình 0.1, phần mở đầu). Ở bước lặp t, thuật giải di truyền duy trì một quần thể các lời giải (các nhiễm sắc thể, các vectơ), $P(t) = \{x_1^t, \ldots, x_n^t\}$. Mỗi lời giải x_i^t được lượng giá để biết được độ "thích nghi" của nó. Rồi một quần thể mới (lần lặp thứ $t+1$) được hình thành bằng cách chọn giữ lại những cá thể thích nghi nhất. Một số cá thể của quần thể này trải qua những biến đổi nhờ *lai tạo (phép lai)* và *đột biến (phép đột biến)*, hình thành nên những lời giải mới. *Phép Lai* kết hợp các tính chất của hai nhiễm sắc thể 'cha' và 'mẹ' để tạo ra các nhiễm sắc thể 'con' bằng cách hoán vị các đoạn gen tương ứng của cha và mẹ. Thí dụ, nếu cha mẹ được biểu diễn bằng vectơ 5 chiều (a_1,b_1,c_1,d_1,e_1) và (a_1,b_2,c_2,d_2,e_2), thì lai tạo, hoán vị tại vị trí thứ 2, sẽ sinh ra các nhiễm sắc thể con (a_1,b_1,c_2,d_2,e_2) và (a_2,b_2,c_1,d_1,e_1). Phép lai cho phép trao đổi thông tin giữa các lời giải.

Khác với phép lai, phép đột biến thay đổi một cách ngẫu nhiên một hay nhiều gen của nhiễm sắc thể được chọn, thay đổi này được thực hiện với một xác suất thể hiện tốc độ đột biến. Phép đột biến cho phép đưa thêm các thông tin mới vào quần thể làm cho chất liệu di truyền phong phú thêm.

Một thuật giải di truyền (hay một chương trình tiến hóa bất kỳ) giải một bài toán cụ thể phải gồm năm thành phần sau đây:

- Cách biểu diễn di truyền cho lời giải của bài toán;
- Cách khởi tạo quần thể ban đầu;
- Một hàm lượng giá đóng vai trò môi trường, đánh giá các lời giải theo mức độ "thích nghi" của chúng;
- Các phép toán di truyền;

- Các tham số khác (kích thước quần thể, xác suất áp dụng các phép toán di truyền vv...)

Để dễ hình dung, chúng tôi sẽ thảo luận các tính năng chính của thuật giải di truyền qua ba thí dụ cụ thể. Trong thí dụ thứ nhất, ta áp dụng thuật giải di truyền tìm giá trị lớn nhất của một hàm thực một biến. Thí dụ thứ hai minh họa cách dùng một thuật giải di truyền để học một chiến lược của một trò chơi đơn giản. Thí dụ 3 bàn về một ứng dụng của thuật giải di truyền để tiếp cận một bài toán tổ hợp NP-đầy đủ, bài toán người du lịch.

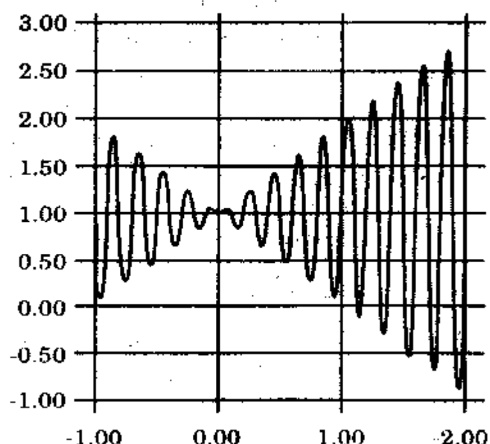
1.1. Tối ưu hàm một biến

Xét bài toán tối ưu không ràng buộc sau:

$Max\ f(x) = x * \sin(10\pi * x) + 1.0;\ x \in [-1,2].$

Hình 1.1 là đồ thị của *f*. Bài toán có nghĩa là tìm *x* trong khoảng [-1,2] để *f* có giá trị lớn nhất, nghĩa là tìm x_o sao cho:

$f(x_0) \geq f(x),\ \forall x \in [-1,2]$



Hình 1.1. Đồ thị hàm $f(x) = x \times \sin(10\pi \times x) + 1.0$

Khi đạo hàm bậc nhất bằng 0, nghĩa là,

$f'(x) = \sin(10\pi * x) + 10\pi x * \cos(10\pi * x) = 0$

$\Leftrightarrow tang\ (10\pi * x) = -10\pi x.$

Rõ ràng là phương trình trên có vô số lời giải,

$$x_i = \frac{2i-1}{20} + \varepsilon_i,\ I = 1, 2, ..$$

$$x_0 = 0$$

$$x_i = \frac{2i+1}{20} - \varepsilon_i\ i = -1, -2, ...$$

trong đó các số hạng ε_i là các dãy số thực giảm (i = 1, 2, ..., và *i*= −1, −2, ...) dần về 0.

Cũng chú ý rằng hàm *f* đạt đến cực đại (cục bộ) tại điểm x_i, khi *i* là số nguyên lẻ, và đạt đến cực tiểu của nó tại x_i, khi *i* chẵn (xem hình 1.1).

Vì miền giá trị của bài toán là [-1,2], hàm đạt cực đại tại $x_{19} = \frac{37}{20} + \varepsilon_{19} = 1.85 + \varepsilon_{19}$, ở đây $f(x_{19})$ hơi lớn hơn $f(1.85) = 1.85 * \sin(18\pi + \pi/2) + 1.0 = 2.85.$

Bây giờ ta dùng thuật giải di truyền để giải bài toán trên, nghĩa là, tìm một điểm trong đoạn [-1, 2] sao cho tại đó *f* có giá trị lớn nhất.

Ta sẽ lần lượt bàn về 5 thành phần chính của thuật giải di truyền giải bài toán này.

1.1.1. Biểu diễn

Ta sử dụng một vectơ nhị phân làm nhiễm sắc thể để biểu diễn các giá trị thực của biến *x*. Chiều dài vectơ phụ thuộc vào độ chính xác cần có, trong thí dụ này, ta tính chính xác đến 6 số lẻ.

Miền giá trị của *x* có chiều dài 2 – (– 1) = 3; với yêu cầu về độ chính xác 6 số lẻ như thế phải chia khoảng [–1, 2] thành ít nhất 3×10^6 khoảng có kích thước bằng nhau. Điều này có nghĩa là cần có 22 bit cho vectơ nhị phân (nhiễm sắc thể):

$$2097152 = 2^{21} < 3000000 \le 2^{22} = 4194304$$

Ánh xạ biến chuỗi nhị phân $(b_{21}b_{20}...b_0)$ thành số thực *x* trong khoảng [–1, 2] được thực hiện qua hai bước như sau:

- đổi chuỗi nhị phân $(b_{21}b_{20}...b_0)$ từ cơ số 2 sang cơ số 10:

$$(< b_{21}b_{20}...b_0 >)_2 = \left(\sum_{i=0}^{21} b_i 2^i\right)_{10} = x'$$

- tìm số thực *x* tương ứng

$$x = -1 + x' \cdot \frac{3}{2^{22} - 1}$$

với –1 là cận dưới của miền giá trị và 3 là chiều dài của miền.

Thí dụ, nhiễm sắc thể (1000101110110101000111) biểu diễn số 0.637197 vì

$x' = (1000101110110101000111)_2 = 2288967_{10}$

và x = –1.0 + 2288967 × 3/4194303 = 0.637197

Đương nhiên nhiễm sắc thể

(0000000000000000000000) và (1111111111111111111111) biểu diễn các cận của miền, -1.0 và 2.0 cho mỗi cận.

1.1.2. Khởi tạo quần thể

Tiến trình khởi tạo rất đơn giản: Ta tạo một quần thể các nhiễm sắc thể, trong đó mỗi nhiễm sắc thể là một vectơ nhị phân 22 bit, tất cả 22 bit của mỗi nhiễm sắc thể đều được khởi tạo ngẫu nhiên.

1.1.3. Hàm lượng giá

Hàm lượng giá *eval* của các vectơ nhị phân *v* chính là hàm *f* :

eval(v) =f(x)

trong đó, nhiễm sắc thể *v* biểu diễn giá trị thực *x* như đã nói ở trên, hàm lượng giá đóng vai trò môi trường, đánh giá từng lời giải theo độ thích nghi của chúng. Thí dụ, 3 nhiễm sắc thể:

v_1= (1000101110110101000111),

v_2 =(0000001110000000010000),

v_3= (1110000000111111000101),

tương ứng với các giá trị x_1 = 0.637197 , x_2 = –0. 958973, và x_3 = 1.627888. Và có độ thích nghi tương ứng:

$eval(v_1) = f(x_1) = 1.586345$,

$eval(v_2) = f(x_2) = 0.078878$,

$eval(v_3) = f(x_3) = 2.250650$.

Rõ ràng, nhiễm sắc thể v_3 là tốt nhất trong 3 nhiễm sắc thể này, vì hàm lượng giá nó trả về giá trị cao nhất.

1.1.4. Các phép toán di truyền

Trong giai đoạn tiến hoá quần thể, ta có thể dùng 2 phép toán di truyền cổ điển: *đột biến* và *lai*.

Như đã trình bày ở trên, đột biến làm thay đổi một (số) gen (các vị trí trong một nhiễm sắc thể) với xác suất bằng tốc độ đột biến. Giả định rằng gen thứ 5 trong nhiễm sắc thể v_3 được chọn để đột biến. Và đột biến chính là thay đổi giá trị gen này: 0 thành 1 và 1 thành 0. Như vậy, sau đột biến này, v_3 sẽ là:

v'_3= (111010000011111111000101)

Nhiễm sắc thể này biểu diễn giá trị x'_3= 1.721638 và $f(x'_3)$= -0.082257. Điều này có nghĩa là đột biến cụ thể này làm giảm khá nhiều giá trị của nhiễm sắc thể v_3 . Bây giờ, nếu gen thứ 10 được chọn để đột biến trong nhiễm sắc thể v_3 thì

v''_3 = (1110000000111111000101) .

Giá trị tương ứng x''_3=1.630818 và $f(x''_3)$= 2.343555, khá hơn giá trị $f(x_3)$ = 2.250650.

Ta sẽ minh họa phép lai trên các nhiễm sắc thể v_2 và v_3. Giả định rằng điểm lai được chọn (ngẫu nhiên) ở vị trí thứ 6:

v_2 = (00000| 1110000000010000),

v_3 = (11100| 00000111111000101).

Hai con của kết quả lai là

v'_2= (00000| 00000111111000101)

v'_3= (11100| 01110000000010000).

Các con này có độ thích nghi:

$f(v'_2)$ =f(-0.998113) = 0.940865,

$f(v'_3)$ =f(1.666028) = 2.459245

Chú ý rằng con thứ 2 thích nghi hơn cả cha lẫn mẹ của nó.

1.1.5. Các tham số

Đối với bài toán đặc biệt này, ta đã dùng các tham số sau đây: kích thước quần thể *pop-size* = 50, xác suất lai tạo p_c =0.25, xác suất đột biến p_m= 0.01. Xác suất lại p_c = 0.25 nghĩa là cá thể v trong quần thể có 25% cơ hội được chọn để thực hiện phép lai; còn xác suất đột biến p_m = 0.01 lại là 1% 1 bít tất bất kỳ của 1 cá thể bất kỳ trong quần thể bị đột biến.

1.1.6. Các kết quả thử nghiệm

Bảng 1.1 trình bày một số kết quả hàm mục tiêu f ở 1 số thế hệ. Cột bên trái cho biết thế hệ được xem xét, và cột bên phải cho biết giá trị của hàm f. Nhiễm sắc thể tốt nhất sau 150 thế hệ là

v_{max} = (1110011010001000000101),

tương ứng với giá trị x_{max} = 1.850773.

Đúng như ta mong đợi, $x_{max} = 1.85 + \varepsilon$, và $f(x_{max})$ lớn hơn 2.85 một chút.

Bảng 1.1. Kết quả của 150 thế hệ

Thế hệ thứ	Hàm lượng giá
1	1.441942
6	2.250003
8	2.250283
9	2.250284
10	2.250363
12	2.328077
39	2.344251
40	2.345087
51	2.738930
99	2.849246
137	2.850217
145	2.850227

1.2. Thế tiến thoái lưỡng nan của tù nhân

Ví dụ này minh họa cách áp dụng thuật giải di truyền để học chiến lược của một trò chơi đơn giản: trò tiến thoái lưỡng nan của tù nhân. Bài toán được mô tả như sau,

Hai tù nhân bị giam trong hai xà lim riêng biệt, không thể thông tin cho nhau. Mỗi tù nhân được yêu cầu là hãy ly khai và phản bội người tù kia, một cách độc lập. Nếu chỉ có một người ly khai, anh ta được thưởng, còn người kia bị phạt. Nếu cả hai đều ly

khai, cả hai đều bị giam lại và tra tấn. Nếu không ai ly khai, cả hai đều được phần thưởng khá. Như vậy, cách chọn ích kỷ là ly khai sẽ luôn đem lại phần thưởng hoặc hình phạt cao hơn, bất chấp người kia chọn ra sao - nhưng nếu cả hai đều ly khai thì họ đều làm những điều tệ hại hơn là hợp tác với nhau. Tình thế khó khăn của người tù là phải quyết định nên ly khai hay hợp tác với người kia.

Trò chơi này gồm hai người chơi, trong trò chơi này, tới phiên mình, mỗi người sẽ chọn ly khai hoặc hợp tác với người kia. Các đấu thủ sẽ được ghi điểm theo cách thưởng phạt được liệt kê trong bảng 1.2.

Bảng 1.2. Bảng thưởng phạt của trò chơi "Tiến thoái lưỡng nan của tù nhân" : P_i là sự thưởng phạt dành cho người chơi *I*.

Đấu thủ 1	Đấu thủ 2	P_1	P_2	Lời bình
Ly khai	Ly khai	1	1	Phạt vì ly khai lẫn nhau
Ly khai	Hợp tác	5	0	Cám dỗ ly khai và sự thưởng phạt
Hợp tác	Ly khai	0	5	Sự thưởng phạt và cám dỗ ly khai
Hợp tác	Hợp tác	3	3	Phần thưởng cho sự hợp tác

Ta sẽ xem cách thuật giải di truyền được sử dụng để học chiến lược của trò chơi tiến thoái lưỡng nan này. Tiếp cận bằng GA nhằm duy trì quần thể các "đấu thủ", mỗi cá thể có chiến lược của riêng nó. Ban đầu, chiến lược của mỗi đấu thủ được chọn ngẫu nhiên. Sau đó, ở mỗi bước, mỗi đấu thủ chơi và được ghi điểm. Rồi có một số đấu thủ được chọn cho thế hệ mới, và một số trong đó được chọn để ghép đôi. Khi hai đấu thủ được ghép đôi, đấu thủ mới được tạo ra có chiến lược

thừa hưởng từ những chiến lược của cha mẹ (lai tạo). Một đột biến, như thường lệ, dẫn tới có biến đổi trong chiến lược của các đối thủ do các thay đổi ngẫu nhiên trên biểu diễn của các chiến lược này.

1.2.1. Biểu diễn chiến lược

Trước hết, ta cần một số cách biểu diễn về chiến lược (nghĩa là, một lời giải khả thi). Để đơn giản, ta sẽ quan tâm đến các chiến lược tất định và sử dụng kết quả của chuyển dịch của 3 bước trước để quyết định bước hiện tại. Vì có 4 khả năng có thể có cho mỗi bước, nên tất cả có $4 \times 4 \times 4 = 64$ khả năng chuyển dịch của 3 bước trước đó.

Có thể đặc tả chiến lược theo kiểu này bằng cách chỉ định chuyển dịch nào cần được thực hiện cho các khả năng có thể xảy ra. Như thế, một chiến lược có thể được biểu diễn bằng chuỗi 64 bit (mỗi bít nhận giá trị Ds -ly khai- hoặc Cs -hợp tác-), chỉ định cho chuyển dịch nào phải làm đối với 64 khả năng. Để có chiến lược khởi điểm ở đầu một chuyển dịch, chúng ta cũng cần đặc tả tiền đề về 3 chuyển dịch giả định trước khi bắt đầu trò chơi. Điều này đòi hỏi phải có hơn 6 gen, như vậy toàn bộ nhiễm sắc thể ít nhất cần 70 bít.

Chuỗi 70 bit đặc tả những gì một đấu thủ trong từng trường hợp có thể xảy ra và như thế xác định hoàn toàn một chiến lược riêng. Chuỗi 70 gen chính là một nhiễm sắc thể của đấu thủ để dùng trong tiến trình tiến hóa.

1.2.2. Thuật giải di truyền

Thuật giải di truyền dùng để học chiến lược tiến thoái lưỡng nan của tù nhân gồm 4 giai đoạn, như sau:

1. *Chọn quần thể khởi đầu.* Mỗi đấu thủ được gán ngẫu nhiên một chuỗi 70 bit, biểu diễn cho một chiến lược như đã trình bày ở trên.

2. *Kiểm tra mỗi đấu thủ để quyết định hiệu quả đạt được.* Mỗi đấu thủ dùng một chiến lược được xác định trong nhiễm sắc thể của nó để đấu với những đấu thủ khác. Điểm của đấu thủ là điểm trung bình của tất cả các bước.
3. *Chọn các đấu thủ để phát sinh.* Nếu đấu thủ có điểm trung bình sẽ được cho một điểm cộng. Khi điểm có độ lệch chuẩn trên trung bình sẽ được cho hai điểm cộng; còn khi điểm có độ lệch chuẩn dưới trung bình sẽ không được điểm cộng nào.
4. *Những đấu thủ chơi thành công sẽ được kết đôi một cách ngẫu nhiên để sinh con qua mỗi lần ghép đôi.* Chiến lược của mỗi con sẽ được quyết định từ những chiến lược của cha mẹ. Điều này được thực hiện bằng cách sử dụng hai phép toán di truyền: *lai* và *đột biến* như trong ví dụ trước.

Sau 4 giai đoạn này ta có một quần thể mới. Quần thể này sẽ biểu diễn các mẫu ứng xử giống cách cư xử của những cá thể thành công trong thế hệ trước nhiều hơn, ít giống cách cư xử của những cá thể không thành công hơn. Với mỗi thế hệ mới, những cá thể có điểm tương đối cao sẽ có nhiều khả năng truyền các phần chiến lược của chúng, trong khi những cá thể không thành công sẽ không có phần chiến lược nào được truyền lại cho thế hệ sau.

1.2.3. Kết quả thực nghiệm

Khi chạy chương trình này, ta nhận được những kết quả rất đáng lưu ý. Từ một khởi đầu rất ngẫu nhiên, thuật giải di truyền làm tiến hóa các quần thể mà thành viên trung gian cũng thành công như thuật giải dùng các heuristic sau:

1. *Đừng lắc lư thuyền*: tiếp tục hợp tác sau 3 lần tương tác (nghĩa là, C sau (CC)(CC)(CC)).
2. *Khiêu khích*: ly khai khi đấu thủ khác ly khai (nghĩa là, D sau khi nhận (CC)(CC)(CD)).

3. *Nhận lời xin lỗi*: tiếp tục hợp tác sau khi việc hợp tác được phục hồi (nghĩa là, C sau (CD)(DC)(CC)).
4. *Quên*: hợp tác khi sự tương tác đã được phục hồi sau khi bị lợi dụng (nghĩa là, C sau (DC)(CC)(CC)).
5. *Nhận ghép đôi*: ly khai sau 3 lần ly khai lẫn nhau. (nghĩa là, D sau (DD)(DD)(DD)).

1.3. Bài toán Người Du Lịch

Ví dụ cuối cùng chúng tôi muốn trình bày là tiếp cận thuật giải di truyền giải bài toán *Người Du Lịch (TSP)*. Bài toán TSP được mô tả như sau.

Một du khách muốn thăm mọi thành phố anh quan tâm; mỗi thành phố thăm qua đúng một lần; rồi trở về điểm khởi hành. Biết trước chi phí di chuyển giữa hai thành phố bất kỳ. Hãy xây dựng một lộ trình thoả các điều kiện trên với tổng chi phí nhỏ nhất.

TSP là bài toán tối ưu tổ hợp và có rất nhiều ứng dụng. Có thể giải bài toán này bằng nhiều phương pháp: phương pháp nhánh cận, phương pháp gần đúng hay những phương pháp tìm kiếm heuristic. Ta sẽ giải bài toán này bằng thuật giải di truyền.

Như ta đã biết, khó khăn đầu tiên đặt ra cho người thiết kế thuật giải di truyền là *biểu diễn nhiễm sắc thể*. Trong ví dụ thứ nhất, tối ưu hàm một biến, bằng cách chấp nhận một ít sai số, ta đã sử dụng biểu diễn nhị phân để biểu diễn nhiễm sắc thể. Còn trong ví dụ thứ hai, với một chút phân tích, ta cũng đưa được về biểu diễn nhị phân các chiến lược cần thiết. Với cách biểu diễn như thế, các phép di truyền truyền thống, lai và đột biến, được áp dụng trực tiếp không cần sửa đổi gì cả. Khi thực hiện lai hay đột biến, các nhiễm sắc thể kết quả vẫn hợp lệ, nghĩa là vẫn là một lời giải thuộc không gian tìm kiếm. Điều này không còn đúng trong bài toán TSP nữa.

Nếu biểu diễn nhị phân cho bài toán TSP có *n* thành phố, mỗi thành phố phải được đánh mã bằng một chuỗi $\lfloor \log_2(n) \rfloor$ bit; và nhiễm sắc thể là một chuỗi gồm $n * \lfloor \log_2(n) \rfloor$ bit. Đột biến có thể tạo ra một lộ trình không thoả điều kiện bài toán nữa: Ta có thể thăm một thành phố 2 lần. Hơn nữa, đối với một bài toán TSP có 20 thành phố (ta cần 5 bit để biểu diễn một thành phố), có một số chuỗi 5 bit nào đó (như 10101) sẽ không tương ứng với thành phố nào cả vì 5 bit có thể biểu diễn tối đa 32 trường hợp. Phép lai cũng gây ra những vấn đề tương tự. Rõ ràng nếu ta dùng các phép toán đột biến và lai truyền thống như đã định nghĩa trước đây, ta sẽ phải dùng đến một loại "thuật giải sửa chữa", thuật giải "sửa chữa" một nhiễm sắc thể không hợp lệ để đưa nó về không gian tìm kiếm.

Cách tự nhiên là ta sẽ đánh số các thành phố và dùng một vectơ nguyên để biểu diễn một nhiễm sắc thể lộ trình. Cách biểu diễn này giúp ta tránh phải dùng thuật giải sửa chữa bằng cách kết hợp những hiểu biết về bài toán vào các phép toán di truyền. Với cách biểu diễn này, một vectơ các thành phần nguyên $v = \langle i_1, i_2, ..., i_n \rangle$ biểu diễn một lộ trình: từ i_1 đến i_2 ..., từ i_{n-1} đến i_n, và trở về i_1 (v là một hoán vị của vectơ <1, 2, ..., n>).

Vấn đề thứ hai là *khởi tạo quần thể đầu*. Đối với tiến trình khởi tạo, ta có thể sử một số thuật giải heuristic (chẳng hạn như dùng thuật giải Greedy áp dụng nguyên lý "tham lam" vào bài toán TSP. Ta sẽ áp dụng thuật giải Greedy nhiều lần, mỗi lần từ một thành phố đầu khác nhau); hoặc đơn giản hơn là khởi tạo quần thể bằng cách tạo ngẫu nhiên các mẫu từ các hoán vị của <1, 2, ..., n>.

Việc lượng giá một nhiễm sắc thể rất dễ dàng: cho trước chi phí của chuyến đi giữa các thành phố, ta có thể dễ dàng tính tổng chi phí của trọn lộ trình.

Về các *phép toán di truyền*, trước hết, ta xây dựng phép *lai OX* như sau: cho trước hai cá thể cha-mẹ, cá thể con có được bằng cách

chọn thứ tự lộ trình từ một cá thể và bảo toàn thứ tự tương đối giữa các thành phố trong cá thể kia. Thí dụ, nếu cha-mẹ là:

<1 2 3 4 5 6 7 8 9 10 11 12> và

<7 3 1 11 4 12 5 2 10 9 6 8>

và khúc được chọn là (4 5 6 7), cá thể con của phép lai sẽ là:

<1 11 12 4 5 6 7 2 10 9 8 3>

Như yêu cầu, cá thể con có quan hệ cấu trúc đối với cả hai cá thể cha-mẹ. Vai trò của cha và mẹ có thể hoán đổi khi xây dựng cá thể con thứ hai.

Phép đột biến thực hiện đơn giản hơn: ta chỉ cần hoán vị hai vị trí bất kỳ trong cá thể được chọn.

(Dĩ nhiên đây chỉ là một cách định nghĩa các phép toán di truyền. Bạn đọc có thể nghĩ ra những cách định nghĩa khác hay hơn).

Như thế, nếu cho trước các tham số liên quan, thuật giải di truyền có thể được thi hành.

Những vấn đề về biểu diễn và các phép toán di truyền giải bài toán TSP sẽ được bàn chi tiết hơn trong phần 3 – Tối ưu tổ hợp.

1.4. Thuật giải leo đồi, mô phỏng việc luyện thép và di truyền

Trong phần này ta bàn về ba thuật giải, đó là leo đồi, mô phỏng việc luyện thép và thuật giải di truyền; cùng áp dụng giải bài toán

tối ưu đơn giản. Thí dụ sau đây chứng tỏ khả năng ưu việt của cách tiếp cận GA.

Không gian tìm kiếm là một tập các chuỗi nhị phân v có chiều dài 30. Hàm mục tiêu f là cần cực đại hóa là hàm:

$f(v) = |11 * \text{one}(v) - 150|$,

trong đó, hàm one(v) cho biết tổng số số 1 có trong chuỗi v. Thí dụ, với ba chuỗi sau đây:

$v_1 = (110110101110101111111011011011)$,

$v_2 = (111000100100110111001010100011)$,

$v_3 = (000010000011001000000010001000)$,

thì

$f(v_1) = |11*22 - 150| = 92$,

$f(v_2) = |11*15 - 150| = 15$,

$f(v_3) = |11*6 - 150| = 84$,

($one(v_1) = 22$; $one(v_2) = 15$ và $one(v_3) = 6$).

f là hàm tuyến tính. Chúng tôi chỉ dùng nó để minh họa ý tưởng chính của ba thuật giải: leo đồi, mô phỏng luyện thép và thuật giải di truyền. Tuy nhiên, cần lưu ý là f có một cực đại toàn cục với

$v_g = (111111111111111111111111111111)$

khi đó $f(v_g) = |11*30 - 150|$ và một cực đại cục bộ khi

$v_l = (000000000000000000000000000000)$

$f(v_l) = |11*0 - 150| = 150$

Có nhiều phiên bản khác nhau của thuật giải leo đồi. Chúng chỉ khác nhau theo cách chuỗi mới được chọn để so sánh với chuỗi hiện tại. Một phiên bản đơn giản (lặp) của leo đồi (*MAX* lần lặp lại) được cho trong hình 1.2. Thoạt đầu, 30 lân cận được quan tâm, và lân cận v_n, trả về giá trị lớn nhất $f(v_n)$, được chọn để cạnh tranh với chuỗi v_c hiện hành. Nếu $f(v_c) < f(v_n)$, thì chuỗi mới trở thành chuỗi hiện hành. Ngược lại thì không có cải thiện cục bộ nào xảy ra: thuật giải đạt đến (cục bộ hoặc toàn cục) tối ưu (cục bộ = TRUE). Trong trường hợp như vậy, lần lặp kế (t ←t+1) được thực hiện với một chuỗi mới, chọn theo ngẫu nhiên.

Đáng chú ý rằng thành công hay thất bại của mỗi lần lặp trong thuật giải leo đồi (nghĩa là trả về tối ưu cục bộ hay toàn cục) tuỳ thuộc chuỗi khởi đầu (được chọn lựa ngẫu nhiên). Rõ ràng là nếu chuỗi khởi đầu có 13 số 1 hoặc ít hơn, thuật giải sẽ luôn luôn chấm dứt tại tối ưu cục bộ (thất bại). Lý do là chuỗi 13 số 1 trả về giá trị 7 của hàm mục tiêu, và bất cứ cải thiện của từng bước lặp đều hướng về tối ưu cục bộ, nghĩa là, tăng số số 1 đến 40, giảm trị số của hàm mục tiêu còn 4. Mặt khác, bất cứ việc giảm lượng số 1 nào cũng sẽ làm tăng trị số của hàm: một chuỗi có 12 số 1 sẽ mang lại giá trị 18 cho hàm, chuỗi 11 số 1 cho trị 29, vv... Điều này có thể đẩy việc tìm kiếm theo hướng "sai", về phía cực đại cục bộ.

Thủ tục leo đồi

bắt đầu

t ← *0*

lặp lại

cục bộ ← FALSE

chọn ngẫu nhiên một chuỗi v_c

tiến hóa v_c

lặp lại

+ chọn 30 chuỗi mới trong lân cận v_c bằng cách thay một các bit của v_c

+ chọn chuỗi v_n trong tập các chuỗi có giá trị hàm mục tiêu *f* lớn nhất

+ nếu $f(v_c) < f(v_n)$

thì $v_c \leftarrow v_n$

nếu không cục bộ ← TRUE

cho đến khi *cục bộ*

t ← *t+1*

cho đến khi *t = MAX*

chấm dứt

Hình 1.2. Thuật giải leo đồi đơn giản (lặp)

Đối với những bài toán có nhiều tối ưu cục bộ, cơ hội chạm được tối ưu toàn cục của thuật giải leo đồi là rất mong manh.

Còn hình 1.3. trang sau trình bày thủ tục mô phỏng luyện thép.

Hàm *random* [0,1) trả về một số ngẫu nhiên trong khoảng [0,1). Các (điều kiện - dừng) kiểm tra xem đã đạt "cân bằng nhiệt độ"

chưa, nghĩa là, phân bố xác suất của các chuỗi mới đã chọn có đạt đến phân bố Boltzmann không. Tuy nhiên, trong một số cài đặt, vòng lặp chỉ được thực thi *k* lần, *k* là tham số bổ sung của phương pháp.

Nhiệt độ *T* được hạ thấp theo từng bước ($g(T,t) < T$ với mọi t). Thuật giải dừng khi *T* đạt một giá trị đủ nhỏ: (tiêu chuẩn - dừng) kiểm tra xem hệ thống đã "đông" lại chưa, nghĩa là, không thay đổi nào được chấp nhận nữa.

Như đã nói trước đây, thuật giải mô phỏng luyện thép có thể thoát khỏi tối ưu cục bộ. Ta hãy xem một chuỗi,

v_4 = (111000000100110111001010100000),

với 12 số 1 cho giá trị của $f(v_4) = |11*12 - 150| = 18$ với v_4 là chuỗi khởi tạo, thuật giải leo đồi có thể tiến đến cực đại cục bộ

v_1 =(000000000000000000000000000000).

Do một chuỗi 30 số 1 bất kỳ (nghĩa là, một bước 'tiến đến' tối ưu toàn cục) cho giá trị 7 (nhỏ hơn 18). Mặt khác, thuật giải mô phỏng luyện thép có thể nhận một chuỗi có 30 số 1 làm chuỗi hiện hành mới với xác suất

$p = \exp\{(f(v_n) - f(v_c))/T\} = \exp\{(7-18)/T\}$,

đối với một nhiệt độ nào đó, chẳng hạn $T= 20$, sẽ cho $p = e^{-11/20} = 0.57695$,

nghĩa là các cơ hội chấp nhận tốt hơn 50%.

Thủ tục mô phỏng luyện thép

bắt đầu

  $t \leftarrow 0$

  khởi tạo nhiệt độ T

  chọn ngẫu nhiên chuỗi v_c làm chuỗi hiện hành

  tiến hóa v_c

  lặp lại

    lặp lại

      + chọn chuỗi v_n mới trong lân cận v_c bằng cách thay một bit của v_c

      + nếu $f(v_c) < f(v_n)$ thì $v_c \leftarrow v_n$

      + nếu không thì nếu random[0,1) < $\exp\{(f(v_n) - f(v_c))/T\}$ thì $v_c \leftarrow v_n$

    cho đến khi (điều kiện dừng thỏa)

    $T \leftarrow g(T,t)$

    $t \leftarrow t+1$

  cho đến khi (điều kiện dừng 2 thỏa)

chấm dứt

Hình 1.3. Thuật giải mô phỏng luyện thép.

Thuật giải di truyền, như đã trình bày trong 1.1, duy trì một quần thể các chuỗi. Hai chuỗi tương đối xấu,

v_5 = (1111100000001101110011101000000) và

v_6 = (0000000000001101110010101111111)

mỗi chuỗi được lượng giá là 16, có thể sinh ra con tốt hơn nhiều (nếu điểm lai rơi vào bất cứ vị trí nào giữa vị trí thứ 5 và thứ 12)

v_7 = (1111100000001101110010101111111).

Con mới được lượng giá

$f(v_7)$ = |11*19 – 150|= 59

Ta kết thúc chương này bằng cách dẫn ra một thông điệp khôi hài vừa được giới thiệu trên Internet (comp.ai.neural-nets).

" Chú ý rằng, trong mọi kỹ thuật [leo đồi] đã thảo luận từ lâu, con thỏ có thể hy vọng tìm được đỉnh núi gần chỗ nó khởi hành. Không có gì bảo đảm đấy là đỉnh Everest, hoặc một ngọn núi rất cao. Nhiều phương pháp khác nhau đã được dùng để thử tìm ra tối ưu toàn cục thực sự.

Trong mô phỏng luyện thép, con thỏ đã được ăn uống và hy vọng một cách ngẫu nhiên suốt một thời gian dài, nó sáng suốt lên và có hy vọng nhảy lên đến đỉnh đồi cao nhất.

Trong thuật giải di truyền, có nhiều thỏ được thả ngẫu nhiên vào một số nơi của dãy Hy mã lạp sơn. Những con thỏ này không biết rằng chúng phải tìm đỉnh Everest. Nhưng, cứ vài năm một lần

bạn phải bắn chúng ở độ cao thấp, với hy vọng những con còn lại sẽ thành công và sinh sản thêm".

1.5. Kết luận

Ba thí dụ về thuật giải di truyền giải bài toán tối ưu hàm, thế tiến thoái lưỡng nan của tù nhân và người du lịch, cho thấy ứng dụng rộng rãi của thuật giải di truyền. Nhưng đồng thời cũng bộc lộ những khó khăn đầu tiên khi sử dụng thuật giải di truyền. Vấn đề biểu diễn cho bài toán người du lịch không rõ ràng lắm. Phép toán mới được sử dụng (lai tạo OX) không hề tầm thường. Còn những khó khăn nào nữa mà ta gặp phải ở những bài toán nào khác (khó hơn)? Trong hai thí dụ, tối ưu hàm và TSP, hàm lượng giá được định nghĩa rõ ràng; trong thí dụ thứ hai (tiến thoái lưỡng nan của tù nhân) một tiến trình mô phỏng đơn giản cho ta một lượng giá của nhiễm sắc thể (ta thử từng đấu thủ để quyết định sự thành công của nó: mỗi đấu thủ dùng một chiến lược do nhiễm sắc thể của nó định nghĩa để chơi với những đấu thủ khác, còn điểm của mỗi đấu thủ là điểm trung bình trên tất cả những trò chơi mà nó chơi). Ta phải tiến hành cách nào trong trường hợp hàm lượng gía không được định nghĩa rõ ràng? Thí dụ, Bài toán Thoả mãn biến Boolean (Boolean Satisfiability Problem - BSP) dường như cần có một biểu diễn chuỗi nhị phân (bit thứ i biểu diễn thứ tự đúng của biến Boolean thứ i), tuy nhiên, tiến trình chọn một hàm lượng giá không phải là rõ ràng (xem các chương sau).

Thí dụ thứ nhất về tối ưu hàm không ràng buộc cho phép ta dùng một biểu diễn tiện lợi, trong đó bất cứ chuỗi nhị phân nào cũng tương ứng với một giá trị trong miền giá trị của bài toán (nghĩa

là, [-1,2]). Điều này có nghĩa là bất cứ một đột biến hay một lai tạo nào cũng sản sinh một con hợp lệ. Điều này cũng đúng trong thí dụ thứ hai: Một kết hợp các bit bất kỳ biểu diễn một chiến lược hợp lệ. Bài toán thứ ba có một ràng buộc duy nhất: mỗi thành phố phải xuất hiện chính xác một lần trong lộ trình hợp lệ. Điều này gây ra vài vấn đề: ta đã dùng các vectơ số nguyên (thay vì biểu diễn nhị phân) và sửa lại phép toán lai. Thế bằng cách nào chúng ta tiếp cận một bài toán có ràng buộc tổng quát ? Chúng ta phải có những khả năng gì ?

Câu trả lời thật không dễ dàng; chúng ta sẽ xem xét các giải pháp ở trong những phần sau.

Chương 2

THUẬT GIẢI DI TRUYỀN : CƠ CHẾ THỰC HIỆN

Trong chương này, chúng tôi sẽ trình bày về cơ chế thực hiện của thuật giải di truyền thông qua một bài toán tối ưu số đơn giản. Ta sẽ bắt đầu thảo luận một vài ý kiến chung trước khi đi vào chi tiết ví dụ.

Không mất tính tổng quát, ta giả định những bài toán tối ưu là bài toán tìm giá trị cực đại. Bài toán tìm cực tiểu hàm f chính là tìm cực đại hàm $g = -f$:

$$\min f(x) = \max g(x) = \max \{-f(x)\}$$

Hơn nữa, ta có thể giả định rằng hàm mục tiêu f có giá trị dương trên miền xác định của nó, nếu không, ta có thể cộng thêm một hằng số C dương, nghĩa là:

$$\max g(x) = \max \{g(x) + C\}$$

(vế trái và vế phải cùng đạt max tại 1 điểm x_0)

Bây giờ, giả sử ta muốn tìm cực đại một hàm k biến $f(x_1,...,x_k)$: $R^k \to R$. Giả sử thêm là mỗi biến x_i có thể nhận giá trị trong miền $D_i = [a_i, b_i] \subseteq R$ và $f(x_1,...,x_k) > 0$ với mọi x_i thuộc D_i. Ta muốn tối ưu hóa hàm f với một độ chính xác cho trước: giả sử cần 6 số lẻ đối với giá trị của các biến.

Rõ ràng là để đạt được độ chính xác như vậy mỗi miền D_i được phân cắt thành $(b_i-a_i)\times10^6$ miền con bằng nhau. Gọi m_i là số nguyên nhỏ nhất sao cho

$$(b_i - a_i) \times 10^6 \le 2^{m_i} - 1.$$

Như vậy, mỗi biến x_i được biểu diễn bằng một chuỗi nhị phân có chiều dài m_i. Biểu diễn như trên, rõ ràng thoả mãn điều kiện về độ chính xác yêu cầu. Công thức sau tính giá trị thập phân của mỗi chuỗi nhị phân biểu diễn biến x_i,

$$x_i = a_i + decimal(111001...001_2).\ \frac{b_i - a_i}{2^{m_i} - 1}$$

trong đó *decimal(chuỗi₂)* cho biết giá trị thập phân của chuỗi nhị phân đó.

Bây giờ, mỗi nhiễm sắc thể (là một lời giải) được biểu diễn bằng chuỗi nhị phân có chiều dài $m=\sum_{i=1}^{k} m_i$; m_1 bit đầu tiên biểu diễn các giá trị trong khoảng $[a_1,b_1]$; m_2 bit kế tiếp biểu diễn giá trị trong khoảng $[a_2, b_2]$; ... ; nhóm m_k bít cuối cùng biểu diễn giá trị trong khoảng $[a_k, b_k]$.

Để khởi tạo quần thể, chỉ cần đơn giản tạo *pop-size* nhiễm sắc thể ngẫu nhiên theo từng bit. Nhưng nếu bạn biết một chút về xác suất, thì nên dùng những hiểu biết về sự phân phối để khởi tạo quần thể ban đầu sẽ tốt hơn.

Phần còn lại của thuật giải di truyền rất đơn giản: trong mỗi thế hệ, ta lượng giá từng nhiễm sắc thể (tính giá trị hàm *f* trên các chuỗi biến nhị phân đã được giải mã), chọn quần thể mới thỏa phân bố xác suất dựa trên độ thích nghi và thực hiện các phép đột biến và lai để tạo các cá thể thế hệ mới. Sau một số thế hệ, khi không còn cải thiện thêm được gì nữa, nhiễm sắc thể tốt nhất sẽ được xem

như lời giải của bài toán tối ưu (thường là toàn cục). Thông thường, ta cho dừng thuật giải di truyền sau một số bước lặp cố định tùy thuộc điều kiện về tốc độ và tài nguyên máy tính.

Đối với tiến trình chọn lọc (chọn quần thể mới thỏa phân bố xác suất dựa trên các độ thích nghi), ta dùng bánh xe quay Ru lét với các rãnh được định kích thước theo độ thích nghi. Ta xây dựng bánh xe Rulét như sau (giả định rằng, các độ thích nghi đều dương, trong trường hợp ngược lại thì ta có thể dùng một vài phép biến đổi tương ứng để định lại tỷ lệ sao cho các độ thích nghi đều dương)

- Tính độ thích nghi $eval(v_i)$ của mỗi nhiễm sắc thể v_i *(i= 1...pop-size)*.
- Tìm tổng giá trị thích nghi toàn quần thể: $F = \sum_{i=1}^{pop_size} eval(v_i)$
- Tính xác suất chọn p_i cho mỗi nhiễm sắc thể v_i, *(i= 1...pop-size)*: $p_i = eval(v_i)/F$
- Tính vị trí xác suất q_i của mỗi nhiễm sắc thể v_i, *(i= 1...pop-size)*. $q_i = \sum_{j=1}^{i} p_j$

Tiến trình chọn lọc được thực hiện bằng cách quay bánh xe ru lét *pop-size* lần; mỗi lần chọn một nhiễm sắc thể từ quần thể hiện hành vào quần thể mới theo cách sau:

- Phát sinh ngẫu nhiên một số *r* trong khoảng [0..1]
- Nếu $r < q_1$ thì chọn nhiễm sắc thể đầu tiên (v_1); ngược lại thì chọn nhiễm sắc thể thứ *i*, v_i ($2 \le i \le pop\text{-}size$) sao cho $q_{i-1} < r \le q_i$.

Hiển nhiên, có thể sẽ có một số nhiễm sắc thể được chọn nhiều lần. Điều này phù hợp với *lý thuyết sơ đồ* (xem chương 3): các nhiễm

sắc thể tốt nhất có nhiều bản sao hơn, các nhiễm sắc thể trung bình không thay đổi, các nhiễm sắc thể kém nhất thì chết đi.

Bây giờ ta có thể áp dụng phép toán di truyền: kết hợp, lai vào các cá thể trong quần thể mới, vừa được chọn từ quần thể cũ như trên. Một trong những tham số của hệ di truyền là xác suất lai p_c. Xác suất này cho ta số nhiễm sắc thể $pop\text{-}size \times p_c$ mong đợi, các nhiễm sắc thể này được dùng trong tác vụ lai tạo. Ta tiến hành theo cách sau đây:

Đối với mỗi nhiễm sắc thể trong quần thể (mới) :

- Phát sinh ngẫu nhiên một một số r trong khoảng [0..1];
- Nếu $r < p_c$, hãy chọn nhiễm sắc thể đó để lai tạo.

Bây giờ, ta ghép đôi các nhiễm sắc thể đã chọn được một cách ngẫu nhiên: đối với mỗi cặp nhiễm sắc thể được ghép đôi, ta phát sinh ngẫu nhiên một số nguyên *pos* trong khoảng [1..m-1] (m là tổng chiều dài - số bit - của một nhiễm sắc thể). Số *pos* cho biết vị trí của điểm lai. Hai nhiễm sắc thể:

$(b_1b_2 \ldots b_{pos}b_{pos+1} \ldots b_m)$ và

$(c_1c_2 \ldots c_{pos}c_{pos+1} \ldots c_m)$

được thay bằng một cặp con của chúng:

$(b_1b_2 \ldots b_{pos}\ c_{pos+1} \ldots c_m)$

$(c_1c_2 \ldots c_{pos}\ b_{pos+11} \ldots b_m)$

Phép toán kế tiếp, đột biến, được thực hiện trên cơ sở từng bit. Một tham số khác của hệ thống di truyền p_m, cho ta số bit đột biến $p_m \times m \times pop\text{-}size$ mong đợi. Mỗi bit (trong tất cả các nhiễm sắc thể

trong quần thể) có cơ hội bị đột biến như nhau, nghĩa là, đổi từ 0 thành 1 hoặc ngược lại. Vì thế ta tiến hành theo cách sau đây.

Đối với mỗi nhiễm sắc thể trong quần thể hiện hành (nghĩa là sau khi lai) và đối với mỗi bit trong nhiễm sắc thể:

- Phát sinh ngẫu nhiên một số r trong khoảng [0..1];
- Nếu $r < p_m$, hãy đột biến bit đó.

Sau quá trình chọn lọc, lai và đột biến, quần thể mới đến lượt lượng giá kế tiếp của nó. Lượng giá này được dùng để xây dựng phân bố xác suất (cho tiến trình chọn lựa kế tiếp), nghĩa là, để xây dựng lại bánh xe rulét với các rãnh được định kích thước theo các giá trị thích nghi hiện hành. Phần còn lại của tiến hóa chỉ là lặp lại chu trình của những bước trên (xem hình 0.1 trong phần dẫn nhập).

Toàn bộ tiến trình sẽ được minh họa trong một thí dụ. Trong ví dụ này, ta sẽ mô phỏng thuật giải di truyền giải bài toán tối ưu số.

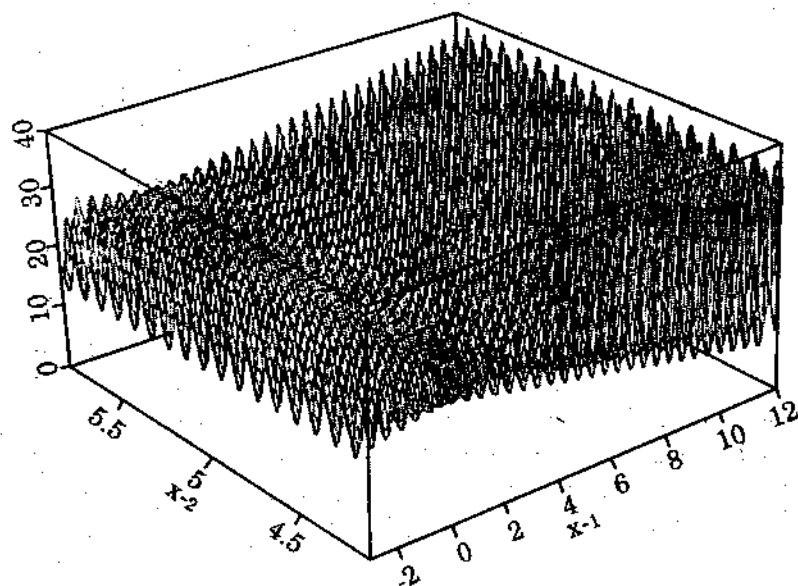
Giả sử kích thước quần thể *pop-size* = 20, và các xác suất di truyền tương ứng là p_c =0.25 và p_m =0.01.

Ta cần cực đại hóa hàm sau đây:

$$f(x_1,x_2) = 21.5 + x_1 \times \sin(4\pi x_1) + x_2 \times \sin(20\pi x_2)$$

với $-3.0 \le x_1 \le 12.1$ và $4.1 \le x_2 \le 5.8$.

Hình 2.1. là đồ thị của hàm f.



Hình 2.1. Đồ thị hàm $f(x_1,x_2) = 21.5 + x_1\times \sin(4\pi x_1) + x_2\times \sin(20\pi x_2)$

Giả sử ta cần tính chính xác đến 4 số lẻ đối với mỗi biến. Miền của biến x_1 có chiều dài 15.1; điều kiện chính xác đòi hỏi đoạn [-3.0,12.1] cần được được chia thành các khoảng có kích thước bằng nhau, ít nhất là 15.1×10000 khoảng. Điều này nghĩa là cần 18 bit làm phần đầu tiên của nhiễm sắc thể:

$$2^{17} \le 151000 \le 2^{18}$$

Miền của biến x_2 có chiều dài 1.7; điều kiện chính xác đòi hỏi đoạn [4.1,5.8] cần được được chia thành các khoảng có kích thước bằng nhau, ít nhất là 1.7×10000 khoảng. Điều này nghĩa là cần 15 bit làm phần đầu tiên của nhiễm sắc thể:

$$2^{14} \le 17000 \le 2^{15}$$

Chiều dài toàn bộ nhiễm sắc thể (vectơ lời giải) lúc này là m=18+15= 33 bit; 18 bit đầu tiên mã hóa x_1, và 15 bit còn lại (từ 19 đến 33) mã hóa x_2.

Ta hãy xét một nhiễm sắc thể làm thí dụ:

(<u>010001001011010000</u>111110010100010)

18 bit đầu tiên, <u>010001001011010000</u>, biểu diễn

$$x_1 = -3.0 + decimal(010001001011010000_2) \times \frac{12.1-(-3.0)}{2^{18}-1}$$

$$= -3,0+70352\,\frac{15.1}{2262143} = -3.0 + 4.052426$$

15 bit kế tiếp 111110010100010, biểu diễn

$$x_2 = 4.1 + decimal(111110010100010_2) \times \frac{5.8-4.1}{2^{15}-1} = 4.1 + 31906.\frac{1.7}{32767}$$

$$= .1+1.655330$$

$$= 5.755330$$

Như vậy, nhiễm sắc thể

(010001001011010000111110010100010)

tương ứng với $<x_1,x_2> = <1.052426, 5.755330>$.

Độ thích nghi của nhiễm sắc thể này là

$f(1.052426, 5.755330)= 20.252640$.

Để cực đại hóa hàm *f* bằng thuật giải di truyền, ta tạo một quần thể có *pop_size* = 20 nhiễm sắc thể. Cả 33 bit trong tất cả các nhiễm sắc thể đều được khởi tạo ngẫu nhiên.

Giả sử rằng sau tiến trình khởi tạo , ta có quần thể sau đây:

v_1= (100110100000001111111010011011111)

v_2= (111000100100110111001010100011010)

v_3= (000010000011001000001010111011101)

v_4= (100011000101101001111000001110010)

v_5= (000111011001010011010111111000101)

v_6= (000101000010010101001010111111011)

v_7= (001000100000110101111011011111011)

v_8= (100001100001110100010110101100111)

v_9= (010000000101100010110000001111100)

v_{10}=(000001111000110000011010000111011)

v_{11}=(011001111110110101100001101111000)

v_{12}=(110100010111101101000101010000000)

v_{13}=(111011111010001000110000001000110)

v_{14}=(010010011000001010100111100101001)

v_{15}=(111011101101110000100011111011110)

v_{16}=(110011110000011111100001101001011)

v_{17}=(011010111111001111010001101111101)

v_{18}=(011101000000001110100111110101101)

v_{19}=(000101010011111111110000110001100)

v_{20}=(101110010110011110011000101111110)

Trong giai đoạn lượng giá ta giải mã từng nhiễm sắc thể và tính các giá trị hàm thích nghi từ các giá trị (x_1, x_2) mới giải mã. Ta có:

$eval(v_1)$ = *f*(6.084492, 5.652242) = 26.019600

$eval(v_2)$ = *f*(10.348434, 4.380264) = 7.580015

$eval(v_3)$ = *f*(-2.516603, 4.390381) = 19.626329

$eval(v_4)$ = *f*(5.278638, 5.593460) = 17.406725

$eval(v_5)$ = *f*(-1.255173, 4.734458) = 25.341160

$eval(v_6)$ = *f*(-1.811725, 4.391937) = 18.100417

$eval(v_7)$ = *f*(-0.991471, 5.680258) = 16.020812

$eval(v_8)$ = *f*(4.910618, 4.703018) = 17.959701

$eval(v_9)$ = *f*(0.795406, 5.381472) = 16.127799

$eval(v_{10})$ = *f*(-2.554851, 4.793707) = 21.278435

$eval(v_{11})$ = *f*(3.130078, 4.996097) = 23.410669



$eval(v_{12}) = f(9.356179, 4.239457) = 15.011619$

$eval(v_{13}) = f(11.134646, 5.378671) = 27.316702$

$eval(v_{14}) = f(1.335944, 5.151378) = 19.876294$

$eval(v_{15}) = f(11.089025, 5.054515) = 30.060205$

$eval(v_{16}) = f(9.211598, 4.993762) = 23.967227$

$eval(v_{17}) = f(3.367514, 4.571343) = 13.696165$

$eval(v_{18}) = f(3.843020, 5.158226) = 15.414128$

$eval(v_{19}) = f(-1.746635, 5.395584) = 20.095903$

$eval(v_{20}) = f(7.935998, 4.757338) = 13.666916$

Rõ ràng nhiễm sắc thể v_{15} mạnh nhất và nhiễm sắc thể v_2 yếu nhất.

Bây giờ ta xây dựng hệ thống kiến trúc bánh xe rulét cho tiến trình chọn lọc. Tổng độ thích nghi của quần thể là:

$$F = \sum_{i=1}^{20} eval(v_i) = 387.776822$$

Xác suất chọn lọc p_i của mỗi nhiễm sắc thể v_i (i=1,..,20) là:

$p_1 = eval(v1) \ / \ F = 0.067099$ $p_2 = eval(v\ ) \ / \ F = 0.019547$

$p_3 = eval(v\ ) \ / \ F = 0.050355$ $p_4 = eval(v\ ) \ / \ F = 0.044889$

$p_5 = eval(v\ ) \ / \ F = 0.065350$ $p_6 = eval(v\ ) \ / \ F = 0.046677$

$p_7 = eval(v\ ) \ / \ F = 0.041315$ $p_8 = eval(v\ ) \ / \ F = 0.046315$



$p_9 = eval(v\ ) \ / \ F = 0.041590$ $p_{10} = eval(v\ ) \ / \ F = 0.054873$

$p_{11} = eval(v\ ) \ / \ F = 0.060372$ $p_{12} = eval(v\ ) \ / \ F = 0.038712$

$p_{13} = eval(v\ ) \ / \ F = 0.070444$ $p_{14} = eval(v\ ) \ / \ F = 0.051257$

$p_{15} = eval(v\ ) \ / \ F = 0.077519$ $p_{16} = eval(v\ ) \ / \ F = 0.061549$

$p_{17} = eval(v\ ) \ / \ F = 0.035320$ $p_{18} = eval(v\ ) \ / \ F = 0.039750$

$p_{19} = eval(v\ ) \ / \ F = 0.051823$ $p_{20} = eval(v\ ) \ / \ F = 0.035244$

Các vị trí xác suất q_i của mỗi nhiễm sắc thể v_i (I=1,..,20) là:

$q_1 = 0.067099$ $q_2 = 0.086647$ $q_3 = 0.137001$

$q_4 = 0.181890$ $q_5 = 0.247240$ $q_6 = 0.293917$

$q_7 = 0.335232$ $q_8 = 0.381546$ $q_9 = 0.423137$

$q_{10} = 0.478009$ $q_{11} = 0.538381$ $q_{12} = 0.577093$

$q_{13} = 0.647537$ $q_{14} = 0.698794$ $q_{15} = 0.776314$

$q_{16} = 0.837863$ $q_{17} = 0.873182$ $q_{18} = 0.812932$

$q_{19} = 0.964756$ $q_{20} = 1.000000$

Bây giờ ta quay bánh xe rulét 20 lần; mỗi lần chọn một nhiễm sắc thể cho quần thể mới. Giả sử thứ tự (ngẫu nhiên) của 20 số trong khoảng [0,1] được phát sinh là

0.513870	0.175741	0.308652	0.534534	0.947628
0.171736	0.702231	0.226431	0.494773	0.424720



0.703899	0.389647	0.277226	0.368071	0.983437
0.005398	0.765682	0.646473	0.767139	0.780237

Số đầu tiên r =0.513870 lớn hơn q_{10} và nhỏ hơn q_{11}, nghĩa là nhiễm sắc thể v_{11} được chọn vào quần thể mới; số thứ hai, 0.175741 lớn hơn q_3 nhỏ hơn q_4, nghĩa là v_4 được chọn cho quần thể mới,vv...

Như vậy, quần thể mới gồm có các nhiễm sắc thể sau:

v'_1 = (011001111110110101100001101111000) (v_{11})

v'_2 = (100011000101101001111000001110010) (v_4)

v'_3 = (001000100000110101111011011111011) (v_7)

v'_4 = (011001111110110101100001101111000) (v_{11})

v'_5 = (000101010011111111110000110001100) (v_{19})

v'_6 = (100011000101101001111000001110010) (v_4)

v'_7 = (111011101101110000100011111011110) (v_{15})

v'_8 = (000111011001010011010111111000101) (v_5)

v'_9 = (011001111110110101100001101111000) (v_{11})

v'_{10} = (000010000011001000001010111011101) (v_3)

v'_{11} = (111011101101110000100011111011110) (v_{15})

v'_{12} = (010000000101100010110000001111100) (v_9)

v'_{13} = (000101000010010101001010111111011) (v_6)



v'_{14} = (100001100001110100010110101100111) (v_8)

v'_{15} = (101110010110011110011000101111110) (v_{20})

v'_{16} = (100110100000001111111010011011111) (v_1)

v'_{17} = (000001111000110000011010000111011) (v_{10})

v'_{18} = (111011111010001000110000001000110) (v_{13})

v'_{19} = (111011101101110000100011111011110) (v_{15})

v'_{20} = (110011110000011111100001101001011) (v_{16})

Bây giờ ta sẽ áp dụng phép toán kết hợp, lai cho những cá thể trong quần thể mới (các vectơ v'_i). Xác suất lai p_c = 0.25 vì thế ta hy vọng (trung bình) 25% nhiễm sắc thể (nghĩa là, 5/20) sẽ tham gia lai tạo. Ta tiến hành theo cách sau: đối với mỗi nhiễm sắc thể trong quần thể (mới) ta phát sinh ngẫu nhiên một số r trong khoảng [0,1]; nếu r < 0.25, ta chọn một nhiễm sắc thể cho trước để lai tạo.

Giả sử thứ tự các số ngẫu nhiên là:

0.822951	0.151932	0.625477	0.314685	0.346901
0.917204	0.519760	0.401154	0.606758	0.785402
0.031523	0.869921	0.166525	0.574520	0.758400
0.581893	0.389248	0.200232	0.355635	0.826927

Điều này có nghĩa là các nhiễm sắc thể v'_2, v'_{11}, v'_{13} và v'_{18} đã được chọn để lai tạo. (Ta đã gặp may: số nhiễm sắc thể được chọn là chẵn, vậy có thể ghép thành cặp một cách dễ dàng. Trường hợp là số lẻ, chúng ta sẽ cộng thêm một nhiễm sắc thể ngoại hoặc lấy bớt một



nhiễm sắc thể - thực hiện điều này một cách ngẫu nhiên thì tốt hơn). Bây giờ ta cho phối ngẫu một cách ngẫu nhiên: tức là, hai nhiễm sắc thể đầu tiên (ví dụ v'_2 và v'_{11}) và cặp kế tiếp (v'_{13} và v'_{18}) được kết cặp. Đối với mỗi cặp trong hai cặp này, ta phát sinh một số nguyên ngẫu nhiên *pos* thuộc khoảng {1, .., 32} (32+1 là tổng chiều dài – số bit – trong một nhiễm sắc thể). Số *pos* cho biết vị trí của điểm lai tạo. Cặp nhiễm sắc thể đầu tiên là:

v'_2 = (100011000| 10110100111100000 1110010)

v'_{11}= (111011101| 10111000010001111011110)

và giả sử số phát sinh *pos* = 9. Các nhiễm sắc thể này bị cắt sau bit thứ 9 và được thay bằng cặp con của chúng:

v''_2 = (100011000| 10111000010001111011110)

v''_{11}= (111011101| 10110100111100000 1110010)

Cặp nhiễm sắc thể thứ hai là:

v'_{13}= (00010100001001010100| 1010111111011)

v'_{18}= (11101111101000100011| 0000001000110)

và số phát sinh *pos* = 20. Các nhiễm sắc thể này được thay bởi một cặp con của chúng:

v''_{13} = (00010100001001010100| 0000001000110)

v''_{18} = (11101111101000100011| 1010111111011)

Cuối cùng, quần thể hiện hành là:

v'_1 = (011001111110110101100001101111000)



v'_2 = (100011000101110000100011111011110)

v'_3 = (001000100000110101111011011111011)

v'_4 = (011001111110110101100001101111000)

v'_5 = (000101010011111111110000110001100)

v'_6 = (100011000101101001111000001110010)

v'_7 = (111011101101110000100011111011110)

v'_8 = (000111011001010011010111111000101)

v'_9 = (011001111110110101100001101111000)

v'_{10} = (000010000011001000001010111011101)

v'_{11} = (111011101101101001111000001110010)

v'_{12} = (010000000101100010110000001111100)

v'_{13} = (000101000010010101000000001000110)

v'_{14} = (100001100001110100010110101100111)

v'_{15} = (101110010110011110011000101111110)

v'_{16} = (100110100000001111111010011011111)

v'_{17} = (000001111000110000011010000111011)

v'_{18} = (111011111010001000111010111111011)

v'_{19} = (111011101101110000100011111011110)

v'_{20} = (110011110000011111100001101001011)

Phép toán kế tiếp, đột biến, được thực hiện trên cơ sở từng bit một. Xác suất đột biến p_m= 0.01, vì thế ta hy vọng (trung bình) 1/100 số bit sẽ qua đột biến. Có 660 bit (*m×pop-size=33×20*) trong



toàn quần thể; ta hy vọng (trung bình) 6.6 đột biến mỗi thế hệ. Mỗi bit có cơ hội đột biến ngang nhau, vì thế, đối với mỗi bit trong quần thể, ta phát sinh ngẫu nhiên một số *r* trong khoảng [0,1]; nếu $r < 0.01$, ta đột biến bit này.

Điều này có nghĩa là ta phải phát sinh 660 số ngẫu nhiên. Giả sử, có 5 trong số 660 số này nhỏ hơn 0.01; vị trí bit và số ngẫu nhiên được trình bày dưới đây:

Vị trí bit	Số ngẫu nhiên
112	0.000213
349	0.009945
418	0.008809
429	0.005425
602	0.002835

- Bảng sau cho biết nhiễm sắc thể vị trí của bit bị đột biến tương ứng với 5 vị trí bit trên.

Vị trí bit	Số nhiễm sắc thể	Số bit trong nhiễm sắc thể
112	4	13
349	11	19
418	13	22
429	13	33
602	19	8

Điều này có nghĩa là 4 nhiễm sắc thể chịu ảnh hưởng của phép toán đột biến: một trong số này là nhiễm sắc thể thứ 13, có hai bit bị thay đổi.



Quần thể cuối cùng được liệt kê ở dưới; các bit đột biến được tô đậm và gạch dưới.

v_1 =(011001111110110101100001101111000)

v_2 =(100011000101110000100011111011110)

v_3 =(001000100000110101111011011111011)

v_4 =(011001111110**0**10101100001101111000)

v_5 =(0001010100111111111110000110001100)

v_6 =(100011000101101001111000001110010)

v_7 =(111011101101110000100011111011110)

v_8 =(000111011001010011010111111000101)

v_9 =(011001111110110101100001101111000)

v_{10} =(000010000011001000001010111011101)

v_{11} =(1110111011011010010**0**11000001110010)

v_{12} =(010000000101100010110000001111100)

v_{13} =(000101000010010101000**1**0000100011**1**)

v_{14} =(100001100001110100010110101100111)

v_{15} =(101110010110011110011000101111110)

v_{16} =(100110100000001111111010011011111)



v_{17} =(000001111000110000011010000111011)

v_{18} =(111011111010001000111010111111011)

v_{19} =(111011100101110000100011111011110)

v_{20} =(110011110000011111100001101001011)

Ta vừa hoàn thành một bước lặp (nghĩa là một thế hệ) của thủ tục di truyền (Hình 0.1 trong phần dẫn nhập). Ta xem xét một chút các kết quả của tiến trình tiến hóa quần thể mới. Trong thời kỳ tiến hóa, ta giải mã từng nhiễm sắc thể và tính các giá trị của hàm thích nghi từ giá trị (x_1, x_2) vừa được giải mã. Ta được:

eval(v_1) = *f*(3.130078, 4.996097) = 23.410669

eval(v_2) = *f*(5.279042, 5.054515) = 18.201083

eval(v_3) = *f*(-0.991471, 5.680258) = 16.020812

eval(v_4) = *f*(3.128235, 4.996097) = 23.412613

eval(v_5) = *f*(-1.746635, 5.395584) =20.095903

eval(v_6) = *f*(5.278638, 5.593460) =17.406725

eval(v_7) = *f*(11.089025, 5.054515) = 30.060205

eval(v_8) = *f*(-1.255173, 4.734458) = 25.341160

eval(v_9) = *f*(3.130078, 4.996097) = 23.410669

eval(v_{10}) = *f*(-2.516603, 4.390381) =19.526329

eval(v_{11}) = *f*(11.088621, 4.743434) =33.351874



eval(v_{12}) = *f*(0.795406, 5.381472) = 16.127799

eval(v_{13}) = *f*(-1.811725, 4.209937) = 22.692462

eval(v_{14}) = *f*(4.910618, 6.703018) = 17.959701

eval(v_{15}) = *f*(7.935998, 4.757388) = 13.666916

eval(v_{16}) = *f*(6.084492, 5.652242) = 26.019600

eval(v_{17}) = *f*(-2.554851, 4.793707) = 21.278435

eval(v_{18}) = *f*(11.134646, 5.666976) = 27.591064

eval(v_{19}) = *f*(11.059532, 5.054515) = 27.608441

eval(v_{20}) = *f*(9.211598, 4.993762) = 23.867227

Chú ý rằng tổng độ thích nghi *F* của quần thể mới là 447.049688 cao hơn tổng độ thích nghi của quần thể trước nhiều (387.776822). Cũng thế, nhiễm sắc thể tốt nhất hiện nay v_{11} có độ thích nghi (33.351874) tốt hơn nhiễm sắc thể tốt nhất v_{15} của quần thể trước (30.060205).

Tiến trình lặp lại lần nữa và các phép toán di truyền được áp dụng, lượng giá thế hệ kế tiếp .v.v... Giả sử sau 1000 thế hệ ta được quần thể:

v_1= (111011110110011011100101010111011)

v_2= (111001100110000100010101010111000)

v_3= (111011110111011011100101010111011)

v_4= (111001100010000110000101010111001)

v_5= (111011110111011011100101010111011)

v_6= (111001100111000100000100010100001)

v_7= (110101100010010010001100010110000)

v_8= (111101100010001010001101010010001)

v_9= (111001100010010010001100010110001)

v_{10}= (111011110111011011100101010111011)

v_{11}= (1101011000000010010001100010110000)

v_{12}= (110101100010010010001100010110001)

v_{13}= (111011110111011011100101010111011)

v_{14}= (111001100110000100000101010111011)

v_{15}= (111001101010111001010100110110001)

v_{16}= (111001100110000101000100010100001)

v_{17}= (111001100110000100000101010111011)

v_{18}= (111001100110000100000101010111001)

v_{19}= (111101100010001010001110000010001)

v_{20}= (111001100110000100000101010111001)

Với độ thích nghi tương ứng là:

$eval(v_1) = f(11.120940, 5.092514) = 30.298543$

$eval(v_2) = f(10.588756, 4.667358) = 26.869724$

$eval(v_3) = f(11.124647, 5.092514\ ) = 30.316575$

$eval(v_1) = f(10.574125, 4{,}242410) = 31.933120$

$eval(v_5) = f(11.124627, 5.092514) = 30.316575$

$eval(v_6) = f(10.588756, 4.214603\ ) = 34.356125$

$eval(v_7) = f(9.631066, 4.427881) = 35.458636$

$eval(v_8) = f(11.518106, 4.452835) = 23.309078$

$eval(v_9) = f(10.574816, 4.427933) = 34.393820$

$eval(v_{10}) = f(11.124627, 5.092514) = 30.316575$

$eval(v_{11}) = f(9.623693, 4.427881) = 35.477938$

$eval(v_{12}) = f(9.631066, 4.427933) = 35.456066$

$eval(v_{13}) = f(11.124627, 5.092514) = 30.316575$

$eval(v_{14}) = f(10.588756, 4.242514) = 32.932098$

$eval(v_{15}) = f(10.606555, 4.653714) = 30.746768$

$eval(v_{16}) = f(10.588814, 4.214603) = 34.359545$

$eval(v_{17}) = f(10.588756, 4.242514) = 32.932098$

$eval(v_{18}) = f(10.588756, 4.242410) = 32.956664$

$eval(v_{19}) = f(11.518106, 4.472757) = 19.669670$

$eval(v_{20}) = f(10.588756, 4.242410) = 32.956664$

Tuy nhiên, nếu ta khảo sát tiến trình trong khi chạy, ta có thể khám phá rằng trong những thế hệ đầu, các giá trị thích nghi của một số nhiễm sắc thể tốt hơn giá trị 35.477938 của nhiễm sắc thể tốt nhất sau 1000 thế hệ. Điều này là do các lỗi tạo mẫu hỗn loạn - trong các chương sau ta sẽ bàn về vấn đề này.

Không khó để theo dõi cá thể tốt nhất trong tiến trình tiến hóa. Thông thường (trong các cài đặt thuật giải di truyền) cá thể tốt "trội hơn cả" được lưu trữ tại một vị trí riêng biệt; bằng cách đó, thuật giải có thể duy trì cá thể tốt nhất tìm được trong suốt quá trình (không chắc là cá thể tốt nhất trong quần thể cuối cùng).

Chương 3

THUẬT GIẢI DI TRUYỀN :

NGUYÊN LÝ HOẠT ĐỘNG

Nền tảng lý thuyết của thuật giải di truyền dựa trên biểu diễn chuỗi nhị phân và lý thuyết sơ đồ. Một sơ đồ là một chuỗi, dài bằng chuỗi nhiễm sắc thể, các thành phần của nó có thể nhận một trong các giá trị trong tập ký tự biểu diễn gen hoặc một ký tự *đại diện* '*'. Sơ đồ biểu diễn một không gian con của không gian tìm kiếm. Không gian con này là tập tất cả các chuỗi trong không gian lời giải mà với mọi vị trí trong chuỗi, giá trị của gen trùng với giá trị của sơ đồ; Ký tự đại diện '*' có thể trùng khớp với bất kỳ ký tự biểu diễn gen nào.

Thí dụ, các chuỗi và sơ đồ có chiều dài 10. Sơ đồ (*111100100) sẽ khớp với hai chuỗi:

{(0111100100), (1111100100)}

và sơ đồ (* 1 * 1 1 0 0 1 0 0) sẽ khớp với 4 chuỗi

{(0101100100), (0111100100), (1101100100), (1111100100)}

Đương nhiên, sơ đồ (1001110001) chỉ khớp với chính nó, và sơ đồ (**********) khớp với tất cả các chuỗi có chiều dài 10. Rõ ràng là mỗi sơ đồ cụ thể có tương ứng 2^r chuỗi, với r là số ký tự đại diện '*' có trong sơ đồ. Mặt khác, mỗi chuỗi chiều dài m sẽ khớp với 2^m sơ đồ. Thí dụ, xét chuỗi (1001110001). Chuỗi này phù hợp với 2^{10} sơ đồ sau:

(1 0 0 1 1 1 0 0 0 1)

(* 0 0 1 1 1 0 0 0 1)

(1 * 0 1 1 1 0 0 0 1)

(1 0 * 1 1 1 0 0 0 1)

:

:

(1 0 0 1 1 1 0 0 0 *)

(* * 0 1 1 1 0 0 0 1)

(* 0 * 1 1 1 0 0 0 1)

:

:

(1 0 0 1 1 1 0 0 * *)

(* * * 1 1 1 0 0 0 1)

:

:

(* * * * * * * * * *),

Một chuỗi chiều dài *m*, sẽ có tối đa 3^m sơ đồ. Trong một quần thể kích thước *n*, có thể có tương ứng từ 2^m đến $n \times 2^m$ sơ đồ khác nhau.

Các sơ đồ khác nhau có những đặc trưng khác nhau. Các đặc trưng này thể hiện qua hai thuộc tính quan trọng: *bậc* và *chiều dài xác định*.

(1) *Bậc* của sơ đồ *S* (ký hiệu là $o(S)$) là số các vị trí 0 và 1 có trong sơ đồ. Đây chính là các vị trí cố định (không phải là những vị trí của ký tự đại diện), trong sơ đồ. Nói cách khác, *bậc* là chiều dài của chuỗi trừ đi số ký tự đại diện. *Bậc* xác định đặc trưng của sơ đồ. Thí dụ, ba sơ đồ chiều dài 10 sau:

S_1 = (* * * 0 0 1 * 1 1 0)

S_2 = (* * * * 0 0 * * 0 *)

S_3 = (1 1 1 0 1 * * 0 0 1)

có bậc tương ứng:

$o(S_1)= 6$; $o(S_2) = 3$; $o(S_3)= 8$;

và S_3 là sơ đồ 'đặc hiệu' nhất.

Khái niệm bậc của sơ đồ giúp cho việc tính xác suất sống còn của sơ đồ do ảnh hưởng của đột biến; Ta sẽ thảo luận chi tiết sau.

(2) *Chiều dài xác định* của sơ đồ *S* (ký hiệu là $\delta(S)$) là khoảng cách giữa hai vị trí cố định ở đầu và cuối. Nó định nghĩa 'độ nén' của thông tin chứa trong một sơ đồ. Thí dụ:

$\delta(S_1)= 10 - 4 = 6$; $\delta(S_2)= 9 - 5= 4$; $\delta(S_3)= 10 - 1 = 9$.

Như vậy, một sơ đồ chỉ có một vị trí cố định duy nhất thì sẽ có chiều dài xác định là 0.

Khái niệm *chiều dài xác định* của sơ đồ giúp tính xác suất sống còn của sơ đồ do ảnh hưởng của phép lai; Ta cũng sẽ thảo luận chi tiết sau.

Như đã thảo luận ở các chương trước, tiến trình mô phỏng tiến hóa của thuật giải di truyền là quá trình lặp gồm có 4 bước:

t ← t+1

chọn P(t) từ P(t-1)

tái kết hợp P(t)

lượng giá P(t)



Bước 1, ($t \leftarrow t+1$), chỉ đơn giản đến số thế hệ tiến hóa; và bước cuối (*lượng giá P(t)*) là lượng giá để tính độ thích nghi của các cá thể trong quần thể hiện hành. Hiện tượng chủ yếu của chu trình tiến hóa xảy ra trong hai bước còn lại của: *chọn lọc* và *tái kết hợp*. Ta sẽ bàn về hiệu quả của hai bước này trên một số sơ đồ cần thiết, biểu diễn trong một quần thể.

Ta bắt đầu bằng bước *chọn lọc*.

Giả sử, quần thể có kích thước pop-size = 20, chiều dài của chuỗi (và cũng là chiều dài của các sơ đồ) là $m = 33$ (như trong thí dụ đã trình bày trong chương trước). Giả sử thêm rằng (ở thế hệ thứ t) quần thể gồm có các chuỗi sau đây:

v_1 = (100110100000001111111010011011111)

v_2 = (111000100100110111001010100011010)

v_3 = (000010000011001000001010111011101)

v_4 = (100011000101101001111000001110010)

v_5 = (000111011001010011010111111000101)

v_6 = (000101000010010101001010111111011)

v_7 = (001000100000110101111011011111011)

v_8 = (100001100001110100010110101100111)

v_9 = (011000000101100010110000001111100)



v_{10} = (000001111000110000011010000111011)

v_{11} = (011001111110110101100001101111000)

v_{12} = (110100010111101101000101010000000)

v_{13} = (111011111010001000110000001000110)

v_{14} = (010010011000001010100111100101001)

v_{15} = (111011101101110000100011111011110)

v_{16} = (110011110000011111100001101001011)

v_{17} = (011010111111001111010001101111101)

v_{18} = (011101000000001110100111110101101)

v_{19} = (000101010011111111110000110001100)

v_{20} = (101110010110011110011000101111110)

(1) Đặt $\xi(S, t)$ là số chuỗi trong quần thể, ở thế hệ thứ t, phù hợp với sơ đồ S. Thí dụ, đối với sơ đồ,

S_0 =(* * * * 1 1 1 *),

thì $\xi(S_0, t) = 3$; vì có 3 chuỗi, v_{13}, v_{15} và v_{16}, phù hợp với sơ đồ S_0. Chú ý rằng bậc của sơ đồ S_0, $o(S_0) = 3$, và chiều dài xác định của nó $\delta(S_0) = 7 - 5 = 2$.



(2) Gọi *eval(S,t)* là độ thích nghi của sơ đồ S ở thế hệ *t*. Giả sử có *p* chuỗi $\{v_{i1},...,v_{ip}\}$ trong quần thể phù hợp với sơ đồ *S* vào thời điểm *t*. Thì:

$$eval(S,t) = \frac{\sum_{j=1}^{p} eval(v_{ij})}{p}.$$

Trong bước *chọn lọc*, một quần thể trung gian được tạo ra gồm *pop-size* = 20 các chuỗi được chọn ra từ quần thể hiện hành. Các chuỗi được chọn dựa và độ thích nghi của nó và được chép vào quần thể thế hệ mới. Như ta đã biết trong chương trước, chuỗi v_i có xác suất được chọn là $p_i = eval(v_i)/F(t)$ ($F(t)$ là tổng thích nghi của toàn quần thể vào thời điểm *t*, $F(t) = \sum_{i=1}^{20} eval(v_i)$).

Sau bước *chọn lọc*, ta có $\xi(S, t+1)$ chuỗi phù hợp với sơ đồ *S*. Do:

(1) Với một chuỗi phù hợp với sơ đồ *S*, trung bình xác suất được chọn của nó là $eval(S,t) / F(t)$,

(2) Ở thế hệ *t*, số chuỗi phù hợp với sơ đồ *S* là $\xi(S, t)$ và,

(3) Chọn trong *pop-size* chuỗi,

vậy:

$\xi(S, t+1) = \xi(S, t) * pop\text{-}size * eval(S,t) / F(t)$.

Với $\overline{F(t)} = F(t)/pop_size$ là độ thích nghi trung bình của quần thể, ta có thể viết lại công thức trên thành:

$$\xi(S, t+1) = \xi(S, t) * eval(S,t) / \overline{F(t)} \quad (3.1)$$

Nói cách khác, số chuỗi trong quần thể tăng bằng với tỉ lệ độ thích nghi của sơ đồ với độ thích nghi trung bình của quần thể. Điều



này có nghĩa là sơ đồ "trên trung bình" sẽ nhận được thêm số chuỗi trong quần thể thế hệ kế tiếp, sơ đồ "dưới trung bình" nhận số chuỗi giảm đi, còn sơ đồ trung bình vẫn giữ nguyên mức.

Hệ quả lâu dài của luật trên cũng rõ ràng. Nếu ta cho rằng sơ đồ *S* vẫn giữ trên trung bình ε % (nghĩa là, $eval(S,t) = \overline{F(t)} + \varepsilon * \overline{F(t)}$, thì:

$$\xi(S, t) = \xi(S, 0)(1+\varepsilon)^t,$$

và $\varepsilon = (eval(S,t) - \overline{F(t)}) / \overline{F(t)}$ ($\varepsilon > 0$, đối với các sơ đồ trên trung bình và $\varepsilon < 0$ đối với các sơ đồ dưới trung bình).

Như vậy, ta có thể nói rằng, chẳng những một sơ đồ "trên trung bình" nhận số chuỗi tăng lên trong thế hệ kế tiếp, mà nó cũng tiếp tục nhận số chuỗi tăng theo lũy thừa trong các thế hệ kế tiếp.

Ta gọi phương trình (3.1) là phương trình tăng trưởng sinh sản của sơ đồ.

Trở về với sơ đồ thí dụ, S_0. Vì có ba chuỗi là v_{13}, v_{15} và v_{16} (vào thời điểm *t*) phù hợp với sơ đồ S_0, độ thích nghi *eval* (S_0) của sơ đồ là:

$eval(S_0,t) = (27.316702 + 30.060205 + 23.867227) / 3 = 27.081378$

đồng thời, độ thích nghi trung bình của toàn quần thể là:

$\overline{F(t)} = \sum_{i=1}^{20} eval(v_i) / pop_size = 387.776822 / 20 = 19.388841$

và tỉ lệ thích nghi của sơ đồ S_0 đối với độ thích nghi trung bình của quần thể là:

$eval(S_0,t) / \overline{F(t)} = 1.396751$

Điều này có nghĩa là, nếu sơ đồ S_0 trên trung bình, thì nó nhận số chuỗi tăng theo lũy thừa trong các thế hệ kế tiếp. Đặc biệt, nếu sơ đồ S_0 vẫn giữ trên trung bình do nhân với hằng số 1.396751, thì, vào thời điểm $t+1$, ta sẽ có 3*1.396751 = 4.19 chuỗi phù hợp với S_0 (nghĩa là 4 hoặc 5), vào thời điểm $t+2$: $3*1.396751^2$ = 5.85 chuỗi như vậy (nghĩa là gần 6 chuỗi), vv... .

Dễ thấy rằng, sơ đồ S_0 như vậy xác định một phần hứa hẹn của không gian tìm kiếm và số mẫu đại diện của nó trong quần thể sẽ tăng theo lũy thừa.

Ta sẽ kiểm tra dự đoán này vào thí dụ của ta, với sơ đồ S_0. Trong quần thể vào thời điểm t, sơ đồ S_0 phù hợp với 3 chuỗi v_{13}, v_{15}, và v_{16}. Trong chương trước, ta đã mô phỏng tiến trình *chọn lọc* sử dụng cùng một quần thể để tạo ra quần thể mới. Quần thể mới gồm các nhiễm sắc thể sau:

v'_1 = (0110011111101101011000011011111000) (v_{11})

v'_2 = (1000010001011010011110000001110010) (v_4)

v'_3 = (0010001000000110101111011011111011) (v_7)

v'_4 = (0110011111101101011000011011111000) (v_{11})

v'_5 = (0001010100111111111110000110001100) (v_{19})

v'_6 = (1000110001011010011110000001110010) (v_4)

v'_7 = (1110111011011100001000111111011110) (v_{15})

v'_8 = (0001110110010100110101111111000101) (v_5)

v'_9 = (0110011111101101011000011011111000) (v_{11})

v'_{10} = (0000100000110010000001010111011101) (v_3)

v'_{11} = (1110111011011100001000111111011110) (v_{15})

v'_{12} = (0100000000101100010110000001111100) (v_9)

v'_{13} = (0001010000100101010010101111111011) (v_6)

v'_{14} = (1000011000011101000101101011001111) (v_8)

v'_{15} = (1011100101100111100110001011111110) (v_{20})

v'_{16} = (111001100110000101000100010100001) (v_1)

v'_{17} = (1110011001100001000001010101111011) (v_{10})

v'_{18} = (1110111111010001000110000001000110) (v_{13})

v'_{19} = (1110111011011100001000111111011110) (v_{15})

v'_{20} = (1100111100000111111100001101001011) (v_{16})

Sơ đồ S_0 bây giờ phù hợp với 5 chuỗi v'_7, v'_{11}, v'_{18}, v'_{19} và v'_{20}.

Tuy nhiên, chỉ riêng phép chọn thì không giới thiệu một điểm mới nào đáng lưu tâm từ không gian tìm kiếm; chọn lọc chỉ sao chép một số chuỗi để hình thành quần thể trung gian. Vì thế, bước thứ hai của chu kỳ tiến hóa *tái kết hợp*, có nhiệm vụ giới thiệu những cá thể mới trong quần thể. Điều này được thực hiện bởi các phép toán di truyền: *lai* và *đột biến*. Ta lần lượt xem xét tác động của hai phép toán này trên một số sơ đồ trong quần thể.

PHÉP LAI

Bắt đầu với phép lai và xét thí dụ sau. Như ta đã biết, một chuỗi trong quần thể, chẳng hạn

v'_{18} = (1110111111010001000110000001000110),

phù hợp với tối đa 2^{33} sơ đồ; cụ thể là chuỗi trên phù hợp với hai sơ đồ sau:

S_0= (* * * * 1 1 1 *) và

S_1= (1 1 1 * 1 0).

Giả sử thêm rằng chuỗi này được chọn để thi hành phép lai (như ở chương 2) và (theo như ví dụ ở chương 2, v'_{18} được lai với v'_{13}) vị trí lai phát sinh tại *pos* = 20. Rõ ràng là sơ đồ S_0 vẫn tồn tại, nghĩa là vẫn còn một con phù hợp với S_0. Lý do là vị trí lai này bảo tồn chuỗi '111' trên các vị trí thứ 5, thứ 6, và thứ 7 của chuỗi trong một đứa con: các chuỗi:

v'_{18} = (11101111101000100011| 0000001000110),

v'_{13} = (00010100001001010100| 1010111111011),

sẽ sinh ra:

v''_{18} = (11101111101000100011| 1010111111011)

v''_{13} = (00010100001001010100| 0000001000110)

Tuy nhiên, sơ đồ S_1 có thể bị phá vỡ: không con nào phù hợp với nó. Lý do là các vị trí cố định '111' ở đầu mẫu và các vị trí cố định '10' ở cuối được đặt vào con khác.

Rõ ràng là *chiều dài xác định* của sơ đồ đóng vai trò quan trọng trong xác suất bị loại bỏ hay tồn tại của sơ đồ. Chú ý rằng, *chiều dài xác định* của sơ đồ S_0 là $\delta(S_0) = 2$, còn *chiều dài xác định* của sơ đồ S_1 là $\delta(S_1) = 32$.

Và, các vị trí lai, trong số *m-1* vị trí, có cơ hội được chọn ngang nhau. Điều này có nghĩa là xác suất bị loại của sơ đồ S là:

$$p_d(S) = \frac{\delta(s)}{m-1},$$

và do đó, xác suất tồn tại là:

$$p_s(S) = 1 - \frac{\delta(s)}{m-1}.$$

Cụ thể, các xác suất tồn tại và bị loại của các sơ đồ S_0 và S_1 là:

$p_d(S_0) = 2/32$; $p_s(S_0) = 30/32$; $p_d(S_1) = 32/32 = 1$; $p_s(S_1) = 0$.

Có một điều quan trọng cần lưu ý là chỉ có một số nhiễm sắc thể trải qua lai và xác suất lai là p_c. Điều này có nghĩa xác suất tồn tại của sơ đồ thực sự sẽ là:

$$p_s(S) = 1 - p_c \cdot \frac{\delta(s)}{m-1}.$$

Ta lại xem xét sơ đồ S_0 của ta, vẫn với thí dụ đang xét (p_c= 0.25):

$p_s(S_0) = 1 - 0.25 * 2/32 = 63/64 = 0.984375$.

Cũng chú ý rằng, ngay cả khi đã chọn một vị trí lai trong số các vị trí cố định trong một sơ đồ, sơ đồ vẫn có cơ may tồn tại. Thí dụ, nếu cả hai chuỗi v'_{18} và v'_{13} đều bắt đầu với '111' và tận cùng bằng '10', thì sơ đồ S_1 vẫn tồn tại (nhưng, xác suất của hiện tượng này rất nhỏ). Do đó, ta nên hiệu chỉnh công thức xác suất tồn tại của sơ đồ:

$$p_s(S) \geq 1 - p_c \cdot \frac{\delta(s)}{m-1}$$

Như vậy, tác động kết hợp của chọn lọc và lai cho ta một dạng mới của phương trình tăng trưởng của sơ đồ sinh sản:

$$\xi(S, t+1) = \xi(S,t).eval\ (S,t)\ /\ \overline{F(t)}\left[1 - p_c \cdot \frac{\delta(S)}{m-1}\right] \quad (3.2)$$

Phương trình (3.2) cho biết kỳ vọng số chuỗi phù hợp với sơ đồ S trong thế hệ kế tiếp là hàm của số chuỗi đúng của sơ đồ, về độ thích nghi tương đối của sơ đồ và *chiều dài xác định* của nó. Rõ ràng là sơ đồ trên trung bình có *chiều dài xác định* ngắn vẫn có thể có số chuỗi cá thể khớp với nó và tốc độ tăng theo lũy thừa. Đối với sơ đồ S_0:

$$eval\ (S,t)\ /\ \overline{F(t)}\left[1 - p_c \cdot \frac{\delta(S)}{m-1}\right] = 1.396751 * 0.984375 = 1.374927$$

Điều này có nghĩa là sơ đồ ngắn, trên trung bình S_0 vẫn nhận được số chuỗi tăng theo lũy thừa trong các thế hệ tiếp theo: vào lúc $(t+1)$ ta có 3*1.374927 = 4.12 chuỗi phù hợp với S_0 (chỉ hơi kém hơn 4.19 - là giá trị mà ta chỉ tính đến chọn lọc). Vào lúc $(t+2)$ có : $3*1.374927^2 = 5.67$ chuỗi như vậy (lại chỉ hơi kém 5.85).

ĐỘT BIẾN

Phép toán kế tiếp được bàn đến là đột biến. Phép đột biến thay đổi một vị trí trong nhiễm sắc thể một cách ngẫu nhiên với xác suất p_m. Thay đổi từ 0 thành 1 hay ngược lại. Rõ ràng là tất cả các vị trí cố định của sơ đồ phải giữ không đổi nếu sơ đồ muốn tồn tại qua đột biến. Thí dụ, xét một chuỗi sau trong quần thể, chẳng hạn như chuỗi:

v'_{19} = (111011101101110000100011111011110)

và sơ đồ S_0:

S_0= (* * * * 1 1 1 *)

Giả sử thêm rằng, chuỗi v'_{19} tham gia đột biến, nghĩa là có ít nhất một bit bị đảo, như đã xảy ra trong chương trước. (Nhớ lại

trong chương trước là 4 chuỗi đã trải qua đột biến: một trong những chuỗi này, v'_{13} bị đột biến ở hai vị trí, ba chuỗi kia - kể cả v'_{19} – chỉ đột biến ở một vị trí). Do v'_{19} đột biến tại vị trí thứ 8, nên kết quả của nó:

v''_{19} = (111011100101110000100011111011110),

vẫn phù hợp với sơ đồ S_0. Nếu các vị trí đột biến được chọn là từ 1 đến 4, hay từ 8 đến 33, thì chuỗi kết quả vẫn phù hợp với S_0. Chỉ 3 bit (thứ 5, 6 và 7 - các vị trí bit cố định trong sơ đồ S_0) là "quan trọng": đột biến ít nhất một trong các bit này sẽ loại bỏ sơ đồ S_0. Rõ ràng, số những bit "quan trọng" bằng với *bậc* của sơ đồ, nghĩa là bằng số các vị trí cố định.

Vì xác suất thay đổi của một bit là p_m nên xác suất tồn tại của một bit là 1 - p_m. Một lần đột biến độc lập với các đột biến khác, vì thế xác suất tồn tại của sơ đồ S qua đột biến (nghĩa là chuỗi các đột biến một-bit) là:

$$p_s(S) = (1 - p_m)^{o(S)}$$

Do $p_m \ll 1$, xác suất này có thể được tính gần đúng là:

$$p_s(S) \approx 1 - o\ (S)^*\ p_m$$

Trở lại với sơ đồ thí dụ S_0 và thí dụ đang chạy (p_m= 0.01) :

$$p_s(S_0) \approx 1 - 3 * 0.01 = 0.97$$

Tác động kết hợp của chọn lọc, lai tạo và đột biến cho ta dạng mới của phương trình tăng trưởng sơ đồ sinh sản:

$$\xi(S, t+1) \geq \xi(S,t).eval\ (S,t)\ /\ \overline{F(t)}\left[1 - p_c \cdot \frac{\delta(S)}{m-1} - o(S).p_m\right] \quad (3.3)$$

Cũng như trong dạng đơn giản (phương trình (3.1) và (3.2)), phương trình (3.3) cũng cho ta biết về kỳ vọng số chuỗi phù hợp với sơ đồ S trong thế hệ tiếp theo là hàm theo: số chuỗi phù hợp với sơ đồ, thích nghi tương đối của sơ đồ, *chiều dài xác định* và *bậc* của sơ đồ. Ta lại thấy rằng, các sơ đồ trên trung bình có *chiều dài xác định* ngắn và *bậc*-thấp vẫn có số chuỗi phù hợp với tốc độ tăng theo lũy thừa.

Đối với sơ đồ S_0:

$$eval(S_0,t)/\ \overline{F(t)}\left[1-\rho_c\cdot\frac{\delta(S)}{m-1}-o(S).\rho_m\right] = 1.396751 * 0.954375 = 1.333024$$

Điều này có nghĩa là sơ đồ ngắn, bậc thấp, trên trung bình S_0 vẫn nhận một số chuỗi tăng theo lũy thừa trong các thế hệ kế tiếp. Vào lúc $t+1$, ta có $3 * 1.333024 = 4.00$ chuỗi phù hợp với S_0 (không kém 4.19 nhiều - là giá trị mà chỉ tính đến việc chọn lựa, hay 4.12 - là giá trị mà ta tính đến việc chọn lọc và lai tạo **), vào lúc $t+2$: $3*1.333024^2 = 5.33$ chuỗi như vậy (lại không kém 5.85 hoặc 5.67 nhiều).

Chú ý rằng phương trình (3.3) dựa trên giả định là hàm thích nghi f chỉ trả về các giá trị dương; khi áp dụng GA vào những bài toán tối ưu hóa mà ở đó hàm tối ưu có thể trả về các giá trị âm, ta cần có một số ánh xạ bổ sung giữa các hàm tối ưu và độ thích nghi.

Tóm lại, phương trình tăng trưởng (3.1) cho biết chọn lọc làm tăng tốc độ tạo mẫu của các sơ đồ trên trung bình, và cho biết thay đổi này là theo lũy thừa. Việc tăng trưởng tự nó không đưa ra một sơ đồ mới nào (không được biểu diễn trong tạo mẫu ban đầu, lúc $t=0$). Điều này chính là lý do mà phép lai được đưa vào để giúp việc trao đổi thông tin cấu trúc nhưng ngẫu nhiên. Ngoài ra, phép lai đưa tính biến thiên cao hơn vào quần thể. Tác động kết hợp (gây rối) của những phép này đối với một sơ đồ không hề quan trọng nếu sơ đồ

ngắn và có *bậc* thấp. Kết quả cuối cùng của phương trình tăng trưởng (3.3) có thể được phát biểu như sau:

Định lý1 (Định lý sơ đồ) *Các sơ đồ ngắn, bậc thấp, trên trung bình nhận số chuỗi tăng theo lũy thừa trong các thế hệ tiếp theo của thuật giải di truyền.*

Kết quả tức thời của định lý này là GA khảo sát không gian tìm kiếm bằng những sơ đồ ngắn, *bậc* thấp, do đó những sơ đồ này được dùng để trao đổi thông tin trong khi lai:

Giả thuyết 1 (Giả thuyết khối kiến trúc) *Thuật giải di truyền tìm việc thực hiện gần tối ưu qua việc đặt gần nhau các sơ đồ ngắn, bậc thấp, hiệu quả cao, được gọi là các khối kiến trúc.*

Chúng ta đã thấy thí dụ hoàn hảo về khối kiến trúc trong chương này:

S_0 = (* * * * 1 1 1 *).

S_0 là sơ đồ ngắn, có bậc thấp và cũng trên trung bình (ít nhất là đối với các quần thể ban đầu). Sơ đồ này góp phần lớn trong việc tìm tối ưu.

Dù đã có một số nghiên cứu được thực hiện nhằm chứng minh giả thuyết này, nhưng đối với hầu hết các ứng dụng khó, ta hầu như chỉ tin tưởng vào kết quả thực nghiệm. Suốt 15 năm sau này, nhiều ứng dụng của GA đã được phát triển để hỗ trợ giả thuyết khối kiến trúc trong nhiều lãnh vực khác nhau. Dù vậy, giả thiết này cho thấy là việc biểu diễn nhiễm sắc thể cho thuật giải di truyền có tính quyết định đối với việc thực hiện nó, và việc biểu diễn này cần thỏa mãn ý tưởng về các khối kiến trúc.

Ở phần đầu của chương, chúng ta đã phát biểu là quần thể có *pop-size* cá thể chiều dài m tạo ra ít nhất 2^m và nhiều nhất là *pop-*

*size** 2^m sơ đồ. Một số sơ đồ được xử lý theo cách có lợi: chúng sinh sản theo tốc độ lũy thừa (đáng mong đợi), và không bị loại bỏ bởi lai và đột biến (điều này có thể xảy ra cho các sơ đồ có chiều dài xác định lớn và bậc cao).

Holland đã chứng tỏ cho thấy rằng ít nhất *pop-size* của chúng được xử lý một cách có ích - ông gọi thuộc tính này là *cơ chế song song ẩn*, vì nó đã đạt được mà không cần bất cứ bộ nhớ ngoài hoặc nhu cầu xử lý nào.

Chương này đã trình bày một số giải thích lý thuyết về lý do thành công của thuật giải di truyền. Nhưng hãy chú ý rằng giả thuyết về khối kiến trúc chỉ là vấn đề tin tưởng và dễ bị xâm phạm trong một số bài toán. Thí dụ, giả sử rằng hai sơ đồ ngắn, bậc thấp, (lần này, ta xét các sơ đồ có chiều dài toàn bộ là 11 vị trí),

S_1 = (1 1 1 * * * * * * * *)

S_2= (* * * * * * * * * 1 1),

là trên trung bình, nhưng kết hợp của chúng,

S_3 = (1 1 1 * * * * * * 1 1), thì ít phù hợp hơn

S_4 = (0 0 0 * * * * * * 0 0).

Giả sử thêm rằng chuỗi tối ưu là S_0 = (11111111111) (S_3 phù hợp). Một thuật giải di truyền có thể gặp nhiều khó khăn trong việc hội tụ về S_0, vì nó có thể hội tụ về các điểm như (00011111100). Hiện tượng này được gọi là bị *lừa*: một số khối kiến trúc (các sơ đồ ngắn, bậc thấp) có thể làm lạc hướng thuật giải di truyền và là cho nó hội tụ vào các điểm dưới tối ưu.

Phần 2

TỐI ƯU SỐ

Chương 4

BIỂU DIỄN NHIỄM SẮC THỂ CHO BÀI TOÁN TỐI ƯU SỐ

Khi ứng dụng thuật giải di truyền vào thực tế, đôi khi gặp những bài toán đòi hỏi một cách biểu diễn lời giải thích hợp, nếu không, thuật giải di truyền khó có thể cho lời giải tốt được. Có thể là do GA thường hội tụ sớm về một lời giải tối ưu không - toàn cục; cũng có thể là do đó là bài toán tối ưu với các ràng buộc không tầm thường.

Biểu diễn nhị phân truyền thống có một số bất lợi khi áp dụng GA giải các bài toán số cần độ chính xác cao, trong một không gian số chiều lớn. Thí dụ, tối ưu hàm 100 biến; mỗi biến nhận giá trị trong khoảng [-500, 500], chính xác đến 6 số lẻ, thì chiều dài của vectơ lời giải nhị phân phải là 3000; phát sinh một không gian tìm kiếm khoảng 10^{1000} phần tử. Tìm kiếm trong một không gian như thế, thuật giải di truyền thực hiện rất kém hiệu quả.

Cách biểu diễn nhị phân khiến cho bất kỳ cách mã hoá nào cũng có cùng số sơ đồ, chính vì vậy, cách dùng chuỗi bít mã hóa lời giải là một trong những công việc chính của người nghiên cứu GA. Cách mã hóa chuỗi nhị phân thường giúp dễ dàng phân tích lý thuyết và cho phép xây dựng các toán tử di truyền "đẹp". Nhưng kết quả của 'cơ chế song song ẩn' thực sự không phụ thuộc vào việc sử dụng các chuỗi bit và có lẽ cũng nên thử nghiệm với tập mẫu tự lớn (thay vì nhị phân chỉ 0 và 1) và những toán tử di truyền (có thể là) mới. Đặc biệt, đối với các bài toán tối ưu số với các biến liên tục, ta có thể thử nghiệm với các gen mã hoá là các số thực cùng với các toán tử "di truyền" đặc biệt cho cách mã hoá số thực này.

Trong chương này, chúng tôi trình bày cách biểu diễn không nhị phân để giải bài toán tối ưu số với số biến đủ lớn. Các gen sẽ nhận giá trị thực, thay vì nhị phân như trong thuật giải di truyền gốc. Mục đích chính là để mở rộng không gian tìm kiếm của GA đến gần không gian thực của bài toán hơn: mở rộng đó buộc các phép toán cũng phải được sửa đổi bằng cách sử dụng một số đặc trưng cụ thể của không gian thực. Thí dụ, biểu diễn này có thuộc tính là hai điểm gần nhau trong không gian biểu diễn cũng phải gần nhau trong không gian bài toán, và ngược lại. Điều này không phải luôn luôn đúng trong cách tiếp cận nhị phân. Tuy nhiên, nếu dùng cách biểu diễn nhị phân nhưng theo nguyên tắc *Gray* ta cũng có thể mở rộng không gian tìm kiếm của GA gần với không gian thực của bài toán.

Thủ tục để chuyển một số nhị phân $\boldsymbol{b} = (b_1,\ldots, b_m)$ thành số mã Gray $\boldsymbol{g} = (g_1,\ldots g_m)$ và ngược lại được trình bày trong hình 4.1; tham số m cho biết số bit trong các biểu diễn này.

Thủ tục chuyển từ nhị phân sang mã Gray

Bắt đầu

$g_1 = p_1$

Cho k = 2 **đến** m **Thực hiện**

$g_k = b_{k-1}$ *XOR* b_k

Kết thúc

Thủ tục chuyển từ mã Gray về biểu diễn nhị phân

Bắt đầu

giá_trị = g_1

b_1 = *giá_trị*

Cho k = 2 **đến** m **Thực hiện**

Nếu g_k = 1 **Thì** *giá_trị* = *NOT giá_trị*

b_k = giá_trị

Hết lặp

Kết thúc

Hình 4.1. Các thủ tục chuyển từ nhị phân sang mã Gray và từ mã Gray về dạng nhị phân.

Bảng 4.1. Liệt kê 16 số nhị phân cùng với các mã Gray tương ứng.

Binary	Gray
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Chú ý rằng biểu diễn Gray có đặc điểm là hai điểm gần nhau bất kỳ trong không gian bài toán chỉ sai khác nhau một bit. Nói cách khác, việc tăng giá trị tham số lên một tương ứng với việc thay đổi chỉ một bit trong mã. Cũng cần chú ý là có nhiều cách để chuyển đổi giữa nhị phân và Gray. Thí dụ (trường hợp $m = 4$), cặp ma trận:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad A^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

cho ra hai biến đổi sau:

$$\boldsymbol{g} = A\boldsymbol{b} \text{ và } \boldsymbol{b} = A^{-1}\boldsymbol{g}$$

trong đó phép nhân là phép nhân modulo 2.

Tuy nhiên, ta dùng biểu diễn chấm động, thay vì nhị phân dạng Gray, vì nó gần với không gian bài toán hơn và cũng cho phép dễ cài đặt hiệu quả các phép toán di truyền. Đã có nhiều nghiên cứu so sánh thực nghiệm các cài đặt (nhị phân và chấm động) trên một số bài toán khác nhau. Trong chương này, chúng tôi minh hoạ những khác biệt giữa biểu diễn nhị phân và chấm động cho một bài toán đặc biệt: bài toán tuyến tính - bậc 2. Bài toán tuyến tính-bậc 2, là trường hợp đặc biệt của bài toán ta sẽ sử dụng trong chương sau, được dùng để minh họa chương trình tiến hoá về tính hội tụ sớm và khả năng tìm kiếm lời giải chính xác.

4.1. Mô tả bài toán

Ta đã chọn bài toán điều khiển động sau:

$$min \quad x_N^2 + \sum_{k=0}^{N-1} (x_k^2 + u_k^2)$$

với $x_{k+1} = x_k + u_k$, k= 0,1, . . . , $N - 1$;

x_0 là trạng thái đầu được cho;

$x_k \in R$ là biến trạng thái;

và $u \in R^N$ là vectơ điều khiển.

Nghiệm giải tích của giá trị tối ưu là:

$J^* = K_0 x_0^2$,

trong đó K_k là nghiệm của phương trình Riccati:

$K_k = 1 + K_{k+1} / (1 + K_{k+1})$ và $K_N = 1$.

Trong các thực nghiệm sau, nhiễm sắc thể sẽ biểu diễn vectơ trạng thái điều khiển $\boldsymbol{u}$. Ta cũng đã thừa nhận miền cố định là (-200, 200) cho mỗi u_i. Đối với các thử nghiệm, ta chọn $x_0 = 100$ và $N = 45$, nghĩa là, nhiễm sắc thể $\boldsymbol{u} = \langle u_0, \ldots, u_{44} \rangle$, có giá trị tối ưu J^* =16180.4.

4.2. Hai cài đặt thử nghiệm

Để so sánh, chúng tôi trình bày cả 2 phiên bản: một phiên bản với biểu diễn nhị phân và phiên bản kia dùng biểu diễn thực.

4.2.1. Phiên bản nhị phân

Trong phiên bản nhị phân, mỗi phần tử của vectơ nhiễm sắc thể được mã hóa bằng cùng số bit. Để giải mã nhanh khi chạy chương trình, mỗi phần tử tương ứng một từ nhớ (nói chung, ta có thể sử dụng nhiều bit hơn 1 từ nhớ, nhưng trường hợp này là trường hợp mở rộng). Theo cách này, các phần tử có thể khai báo kiểu integer, với một số thao tác nhị phân hoá cũng như thập phân hoá

một phần tử. Rồi, mỗi nhiễm sắc thể là một vectơ gồm *N* từ nhớ, bằng với số phần tử của mỗi nhiễm sắc thể (hoặc một bội số của từ nhớ, nếu biểu diễn cần nhiều lần số từ nhớ).

Độ chính xác của cách tiếp cận này phụ thuộc (với miền xác định có kích thước cố định) vào số bit thực sự sử dụng và bằng với $(UB-LB) / (2^n-1)$, trong đó *UB* và *LB* là các cận của miền xác định, còn *n* là số bit trong mỗi phần tử của một nhiễm sắc thể.

4.2.2. Phiên bản thực

Trong phiên bản thực, mỗi vectơ nhiễm sắc thể được mã hóa bằng một vectơ các số thực, có cùng độ dài như vectơ lời giải. Mỗi phần tử buộc phải nằm trong khoảng mong muốn, và các phép toán được thiết kế một cách cẩn thận để bảo toàn yêu cầu này.

Độ chính xác của cách tiếp cận này chỉ phụ thuộc vào máy. Đương nhiên, ta luôn có thể mở rộng độ chính xác của biểu diễn nhị phân bằng cách dùng nhiều bit hơn, nhưng như vậy sẽ làm giảm đáng kể tốc độ của thuật giải (xem phần 4.4).

Ngoài ra, biểu diễn thực còn có khả năng biểu diễn những miền thật lớn (hay trường hợp các miền xác định không biết trước). Mặt khác, biểu diễn nhị phân phải hy sinh độ chính xác khi tăng kích thước miền, với chiều dài nhị phân cố định cho trước. Cũng vậy, trong biểu diễn thực, việc thiết kế các công cụ đặc biệt để xử lý các ràng buộc không tầm thường sẽ dễ dàng hơn nhiều: điều này sẽ được bàn đầy đủ trong chương 6.

4.3. Thử nghiệm

Các thử nghiệm được thực hiện trên cùng một cấu hình máy tính. Tất cả các kết quả trình bày ở đây biểu diễn kết quả trung bình đạt được sau 10 lần chạy độc lập của từng phiên bản trên.

Trong tất cả các thử nghiệm, kích thước quần thể được giữ cố định là 60, và số lần lặp là 20000. Nếu không thay đổi trạng thái, biểu diễn nhị phân được dùng $n = 30$ bit để mã hóa một biến (một phần tử của vectơ lời giải), vậy cần 30 x 45 = 1350 bit cho toàn bộ vectơ điều khiển ***u***.

4.3.1. Đột biến và lai tạo ngẫu nhiên

Trong thực nghiệm này, ta chạy cả hai phiên bản với những phép toán tương đương với các phép toán truyền thống.

PHIÊN BẢN NHỊ PHÂN

Phiên bản nhị phân sử dụng những phép toán đột biến và lai tạo truyền thống, nhưng, để cho giống với những phép toán của phiên bản thực hơn, ta chỉ cho phép lai giữa những phần tử. Xác suất lai được giữ cố định là 0.25, trong khi xác suất đột biến được thay đổi để đạt được tốc độ cập nhật nhiễm sắc thể mong muốn (bảng 4.2).

Bảng 4.2. xác suất cập nhật nhiễm sắc thể đối với tốc độ đột biến.

Phiên bản	Xác suất cập nhật nhiễm sắc thể				
	0.6	0.7	0.8	0.9	0.95
Nhị phân, p_m	0.00047	0.00068	0.00098	0.0015	0.0021
Thực, p_m	0.014	0.02	0.03	0.045	0.061

PHIÊN BẢN THỰC

Phép lai của phiên bản thực hoàn toàn giống với phiên bản nhị phân (các điểm phân cách giữa số chấm động) và được áp dụng với cùng xác suất (0.25). Đột biến, áp dụng cho số chấm động thay vì

cho bit; kết quả của đột biến đó là một trị ngẫu nhiên trong miền [LB, UB].

KẾT QUẢ THỰC NGHIỆM

Bảng 4.3. Kết quả trung bình là hàm của xác suất cập nhật nhiễm sắc thể

Phiên bản	Xác suất cập nhật nhiễm sắc thể					Độ lệch chuẩn
	0.6	0.7	0.8	0.9	0.95	
Nhị phân	42179	46102	29290	52769	30573	31212
Thực	46594	41806	47454	69624	82371	11275

Kết quả (bảng 4.3) cho thấy phiên bản nhị phân có vẻ hơi tốt hơn; nhưng cũng khó mà nói rằng chúng tốt hơn thực sự do tất cả đều còn cách xa lời giải tối ưu thực (16180.4). Hơn nữa, một hiện tượng cũng đáng lưu tâm: phiên bản thực ổn định hơn với độ lệch chuẩn thấp hơn nhiều.

Ngoài ra, nếu để ý rằng thực nghiệm trên không hoàn toàn thích hợp đối với biểu diễn thực; đột biến của nó không 'tự nhiên' bằng đột biến của phiên bản nhị phân.

Để minh họa, ta hãy khảo sát: xác suất để sau khi đột biến một phần tử sẽ rơi trong δ % của một khoảng (400, do khoảng giá trị là [-200, 200]) từ giá trị cũ của nó? Câu trả lời là:

Phiên bản thực: xác suất đó rõ ràng rơi trong khoảng [δ, 2δ]. Thí dụ, cho δ = 0.05, nó sẽ trong khoảng [0.05, 0.1].

Phiên bản nhị phân: ở đây ta cần xét số bit bậc thấp có thể thay đổi một cách an toàn. Giả sử chiều dài phần tử n = 30 và m là chiều dài thay đổi cho phép, m phải thỏa $m \le n + \log_2\delta$. Do m là số nguyên, nên $m = \lfloor n + \log_2\delta \rfloor$ = 25 và xác suất là m/n = 25/30 = 0.833, một số hoàn toàn khác.

Do đó, ta sẽ cố thiết kế một phương pháp để bù lấp những khuyết điểm này.

4.3.2. Đột biến không đồng nhất

Trong thực nghiệm này, ta chạy, ngoài các phép toán được nói đến trong phần 4.3.1, còn có một phép toán đột biến đặc biệt có mục đích vừa cải thiện việc tìm chính xác một phần tử vừa làm giảm bất lợi của đột biến ngẫu nhiên trong phiên bản thực. Ta gọi nó là *đột biến không đồng nhất*; trong hai chương sau ta sẽ bàn đầy đủ về phép toán này.

PHIÊN BẢN THỰC

Phép *đột biến không đồng nhất* được định nghĩa như sau: nếu $s_v^t = <v_1,...,v_m>$ là một nhiễm sắc thể (t là số thế hệ) và phần tử v_k được chọn để đột biến, kết quả là một vectơ $s_v^{t+1} = <v'_1, ..., v'_m>$, trong đó,

$$v'_k = \begin{cases} v_k + \Delta(t, UB - v_k), & \text{nếu chữ số ngẫu nhiên là } 0 \\ v_k - \Delta(t, v_k - LB), & \text{nếu chữ số ngẫu nhiên là } 1 \end{cases}$$

LB và UB là cận trên và cận dưới miền xác định của biến v_k. Hàm $\Delta(t,y)$ trả về một giá trị trong khoảng [0, y] sao cho xác suất của $\Delta(t,y)$, gần với 0, tăng khi t tăng. Tính chất này khiến phép toán này lúc đầu tìm kiếm không gian đồng bộ (khi t nhỏ), và rất cục bộ ở những giai đoạn sau; như vậy sẽ tăng xác suất phát sinh một số mới

gắn với hậu duệ hơn là lựa chọn ngẫu nhiên. Ta đã dùng hàm sau đây:

$$\Delta(t,y) = y.(1 - r^{(1-\frac{t}{T})^b})$$

trong đó r là một số ngẫu nhiên trong khoảng [0..1]. T là số thế hệ cực đại và b là tham số hệ thống xác định mức độ phụ thuộc vào số lần lặp (ta dùng b = 5).

PHIÊN BẢN NHỊ PHÂN

Để thích hợp hơn đối với biểu diễn nhị phân, ta mô hình hóa toán tử đột biến không đồng nhất vào không gian của nó, mặc dù, thực sự, nó được đưa vào chủ yếu để cải thiện đột biến của phiên bản thực. Ở đây, tương tự với phép toán của phiên bản thực, nhưng với v'_k được xác định khác:

v'_k = đột biến(v_k, $\nabla(t,n)$),

trong đó, n = 30 là số bit/phần tử của một nhiễm sắc thể; *độtbiến(v_k,pos)* có nghĩa là: giá trị đột biến của phần tử thứ k trên bit *pos* (0 bit là kém ý nghĩa nhất) và

$$\nabla(t,n) = \begin{cases} \lfloor \Delta(t,n) \rfloor, & \text{nếu chữ số ngẫu nhiên là } 0 \\ \lceil \Delta(t,n) \rceil, & \text{nếu chữ số ngẫu nhiên là } 1 \end{cases}$$

với tham số b của Δ được điều chỉnh thích hợp (ta dùng b= 1.5).

KẾT QUẢ THỰC NGHIỆM

Ta lặp lại các thực nghiệm tương tự các như ở phần 4.3.1, dùng đột biến không đồng nhất được áp dụng cùng tỉ lệ với các đột biến được định nghĩa trước đây.

Bảng 4.4. Kết quả trung bình là hàm xác suất của cập nhật nhiễm sắc thể

Phiên Bản	Xác suất cài đặt nhiễm sắc thể		Độ lệch chuẩn
	0.8	0.9	
Nhị phân	35265	30373	40256
Thực	20561	26164	2133

Bây giờ, phiên bản thực cho thấy hiệu quả trung bình tốt hơn (bảng 4.4). Ngoài ra, các kết quả của phiên bản nhị phân lần này cũng không ổn định hơn. Nhưng, để ý rằng ở đây mặc dù có mức trung bình cao, cài đặt nhị phân đã sinh ra hai kết quả tốt nhất cho lần này (16205 và 16189).

4.3.3. Các phép toán khác

Trong phần này ta quyết định cài đặt và dùng các phép toán bổ sung nếu có thể định nghĩa được dễ dàng trong cả hai không gian biểu diễn.

PHIÊN BẢN NHỊ PHÂN

Ngoài những phép toán đã được mô tả như trên, ta cài đặt phép lai nhiều điểm, và cũng cho phép lai trong số các bit của một phần tử. Phép lai nhiều điểm có xác suất áp dụng vào một phần tử được điều khiển bởi tham số hệ thống (là 0.3).

PHIÊN BẢN THỰC

Ở đây ta cũng cài đặt phép lai nhiều điểm tương tự. Ngoài ra, ta cũng cài đặt phép lai số học một và nhiều điểm; chúng tính trung bình các giá trị của hai phần tử chứ không trao đổi chúng, tại các điểm chọn. Những phép toán đó có đặc điểm là các phần tử của các

nhiễm sắc thể mới vẫn thuộc miền xác định của chúng. Hai chương sau sẽ cung cấp thêm chi tiết về các phép toán này.

KẾT QUẢ THỰC NGHIỆM

Ở đây, phiên bản thực cho thấy một ưu thế nổi bật (bảng 4.5); mặc dù các kết quả tốt nhất không khác nhau bao nhiêu, chỉ phiên bản thực vẫn tiếp tục đạt được chúng.

Bảng 4.5. Kết quả trung bình là hàm xác suất của cập nhật nhiễm sắc thể

Phiên Bản	Xác suất cập nhật nhiễm sắc thể			Độ lệch chuẩn	Tốt nhất
	0.7	0.8	0.9		
Nhị phân	23814	19234	27456	6078	16188.2
Thực	16248	16798	16198	54	16182.1

4.4. Hiệu quả về thời gian

Có nhiều nhận xét cho rằng thời gian chạy GA giải các bài toán không tầm thường quá cao. Trong phần này ta sẽ so sánh hiệu quả thời gian của hai phiên bản. Kết quả được trình bày trong bảng 4.6 là kết quả của các lần chạy ở phần 4.3.3.

Bảng 4.6. Thời gian CPU (*giây*) là hàm của số phần tử.

Phiên bản	Số các phần tử (*N*)				
	5	15	25	35	45
Nhị phân	1080	3123	5137	7177	9221
Thực	184	398	611	823	1072

Bảng 4.6 so sánh thời gian CPU của cả hai phiên bản trên số phần tử trên một nhiễm sắc thể. Phiên bản thực nhanh hơn nhiều ngay cả đối với số bit vừa phải là 30 cho mỗi biến trong cài đặt nhị phân. Đối với các miền lớn và độ chính xác cao, chiều dài toàn bộ của nhiễm sắc thể tăng lên, và khác biệt tương đối cũng tăng lên, như sẽ được trình bày trong bảng 4.7.

4.5. Kết luận

Bảng 4.7. Thời gian CPU (*giây*) là hàm theo số bit của mỗi phần tử; *N* = 45

Phiên Bản	Số bit trên mỗi phần tử nhị phân					
	5	10	20	30	40	50
Nhị phân	4426	5355	7438	9219	10981	12734
Thực	1072 (hằng số)					

Các thực nghiệm trên cho thấy biểu diễn chấm động nhanh hơn, nhất quán hơn. Và cho độ chính xác cao hơn (nhất là với các miền xác định rộng mà mã hóa nhị phân phải cần một chuỗi dài để biểu diễn). Đồng thời kết quả của nó được nâng cao bởi các phép toán đặc biệt để đạt được độ chính xác cao (thậm chí cao hơn trong biểu diễn nhị phân). Ngoài ra, biểu diễn chấm động do bản chất đã gần hơn với không gian bài toán nên dễ thiết kế các phép toán khác kết hợp với tri thức bài toán - đặc tả hơn. Điều này cần thiết cho việc xử lý các ràng buộc bài toán đặc tả, không tầm thường (chương 6).

Những kết luận này giải thích tại sao người sử dụng các kỹ thuật tiến hóa di truyền thích biểu diễn chấm động hơn, như trong: (1) thoải mái với tương quan một biến - một gen, (2) tránh được các dốc Hamming và các thao tác khác trong việc thực hiện đột biến trên các chuỗi bit được xử lý như các số nguyên nhị phân không dấu, (3) có ít thế hệ hơn.

Tuy nhiên, nếu số biến không quá lớn, nhỏ hơn 10, và chấp nhận trước một ít sai số, phiên bản nhị phân thực sự vẫn hiệu quả hơn. Vì thế, bạn nên thực hiện một vài thử nghiệm. Chọn một số hàm thử nghiệm và thử nghiệm với ba hệ thống dựa trên GA với các biểu diễn nhị phân, Gray, và chấm động. Hơn nữa, bạn cũng có thể sử dụng GA kết hợp với leo đồi hay 1 số kỹ thuật khác có thể tăng độ chính xác gần với kết quả thực.

CHƯƠNG 5

BÀN THÊM VỀ PHÉP ĐỘT BIẾN KHÔNG ĐỒNG BỘ

Thuật giải di truyền thể hiện những khó khăn vốn có trong việc thực hiện tìm kiếm cục bộ cho các ứng dụng số. Holland đề nghị rằng thuật giải di truyền nên được dùng như một bộ tiền xử lý, trước khi giao việc xử lý tìm kiếm cho một hệ thống có khả năng sử dụng tri thức về miền, để hướng dẫn việc tìm kiếm cục bộ.

Tìm kiếm cục bộ cần sử dụng các lược đồ bậc cao và chiều dài xác định lớn hơn những gì được đề nghị trong Lý thuyết sơ đồ. Ngoài ra, có những bài toán có miền tham số không bị giới hạn, số biến lớn, và cần độ chính xác cao, những yêu cầu này có nghĩa là chiều dài của vectơ lời giải nhị phân rất lớn (đối với 100 biến, với các miền trong khoảng [-500, 500], cần độ chính xác 6 số lẻ, thì chiều dài của vectơ lời giải nhị phân là 3000). Như đã trình bày trong chương trước, thuật giải di truyền thực hiện những bài toán như thế rất kém hiệu quả.

Để cải thiện khả năng tìm chính xác của thuật giải di truyền - là điều bắt buộc của các bài toán cần độ chính xác cao - ta thiết kế một toán tử đột biến đặc biệt mà hiệu quả của nó khác hẳn đột biến truyền thống. Nhớ lại rằng một đột biến truyền thống thay đổi mỗi lần một bit của nhiễm sắc thể; vì vậy, thay đổi đó chỉ dùng tri thức cục bộ - chỉ biết đến bit đang bị đột biến. Nếu bit đó nằm ở phần bên trái của chuỗi mã hóa một biến, thì tác động đột biến trên biến đó rất có ý nghĩa. Ngược lại, những bit ở tít đầu bên phải của chuỗi lại có tác động hoàn toàn nhỏ khi đột biến. Ta quyết định dùng tri thức toàn cục về vị trí theo cách sau đây: khi quần thể già đi, những bit nằm ở tít bên phải của mỗi chuỗi mã hóa biến có xác suất được

đột biến cao hơn, trong khi những bit nằm trái nhất lại có xác suất giảm. Nói cách khác, một đột biến như thế tạo ra tìm kiếm toàn cục của không gian tìm kiếm lúc bắt đầu tiến trình lặp, nhưng càng về sau thì khai thác cục bộ lại tăng lên. Ta gọi đây là đột biến không đồng nhất và sẽ bàn về nó trong chương này.

Trước tiên, ta nói về các bài toán được dùng thử nghiệm cho toán tử mới này.

5.1. Các trường hợp thử nghiệm

Nói chung, thiết kế và cài đặt thuật giải di truyền giải những bài toán điều khiển tối ưu là rất khó. Trong chương này, chúng tôi thực nghiệm thuật giải di truyền với đột biến không đồng nhất trên ba bài toán điều khiển tối ưu thời gian – rời rạc: bài toán tuyến tính bình phương, bài toán thu hoạch và bài toán xe kéo đã được rời rạc hóa.

5.1.1. Bài toán tuyến tính bình phương

Bài toán thử nghiệm đầu tiên là mô hình tuyến tính bình phương một chiều:

$$\min q.x_N^2 + \sum_{k=0}^{N-1} (s.x_k^2 + r.u_k^2) \quad (5.1)$$

với:

$$x_{k+1} = ax_k + b^* u_k\,,\ \ k = 0,1,\ldots,N\text{-}1 \quad (5.2)$$

Trong đó x_0 cho trước, a, b, q, s, r là các hằng cho trước, $x_k \in R$ là trạng thái còn $u_k \in R$ là điều khiển của hệ thống.

Lời giải chính xác của 5.1 thỏa 5.2 là:

$$J^* = K_0 x_0^2 \ (5.3)$$

ở đây K_k là nghiệm của phương trình Ricati:

$$K_k = s + ra^2 K_k + 1 / (r + b^2 K_{k+1}),\ K_N = q \quad (5.4)$$

Hệ quả là, bài toán (5.1) dẫn tới (5.2) sẽ được giải với các tập tham số trong bảng 5.1.

Trường hợp	N	x_0	s	r	q	a	b
I	45	100	1	1	1	1	1
II	45	100	10	1	1	1	1
III	45	100	100	1	1	1	1
IV	45	100	1	10	1	1	1
V	45	100	1	1000	1	1	1
VI	45	100	1	1	0	1	1
VII	45	100	1	1	1000	1	1
VIII	45	100	1	1	1	0.01	1
IX	45	100	1	1	1	1	0.01
X	45	100	1	1	1	1	100

Trong thử nghiệm giá trị của N là 45

5.1.2. Bài toán thu hoạch

Bài toán thu hoạch được định nghĩa là:

$$\max \sum_{k=0}^{N-1} \sqrt{u_k} \quad (5.5)$$

với phương trình tăng trưởng:

$$x_{k+1} = a^* x_k - u_k \quad (5.6)$$

và một ràng buộc là:

$$x_0 = x_N \quad (5.7)$$

với trạng thái khởi đầu cho trước là x_0 , a là hằng số, và $x_k \in R$, $u_k \in R^+$ theo thứ tự là trạng thái và biến điều khiển (không âm).

Lời giải tối ưu chính xác J^* của bài toán (5.5) thỏa (5.6) và (5.7) là:

$$J^* = \sqrt{\frac{x_0.(a^N - 1)^2}{a^{N-1}.(a-1)}} \quad (5.8)$$

Bài toán (5.5) thỏa (5.6) và (5.7) sẽ được giải với a = 1.1, x_0 = 100, và các giá trị N = 2, 4, 10, 20 và 45.

5.1.3. Bài toán xe kéo

Vấn đề của bài toán xe kéo là cực đại hóa khoảng cách du hành tổng cộng $x_1(N)$ trong một thời gian cho trước (tức là một đơn vị) trừ tổng các cố gắng.

$$x_1(k+1) = x_2(k) \quad (5.9)$$

$$x_2(k+1) = 2x_2(k) - x_1(k) + 1/N^2 * u(k) \quad (5.10)$$

và chỉ số hiệu quả cần cực đại hóa là:

$$x_1(N) - \frac{1}{2N} \sum_{k=o}^{N-1} u^2(k) \quad (5.11)$$

Với bài toán này, giá trị tối ưu của chỉ số trong (5.11) là:

$$J^* = \frac{1}{3} - \frac{3N-1}{6N^2} - \frac{1}{2N^3} \sum_{k=o}^{N-1} k^2$$

Bài toán xe kéo được giải với các giá trị N = 5, 10, 15, 20, 25, 30, 35, 40 và 45.

5.2. Chương trình tiến hóa giải bài toán tối ưu hóa số

Chương trình tiến hóa ta đã xây dựng cho các bài toán tối ưu số được dựa trên biểu diễn thực, và một số toán tử di truyền mới (chuyên biệt hóa); ta sẽ lần lượt bàn về chúng.

5.2.1. Biểu diễn

Trong biểu diễn thực, mỗi vectơ nhiễm sắc thể được mã hóa thành vectơ thực có cùng chiều dài với vectơ lời giải. Mỗi phần tử được chọn lúc khởi tạo sao cho thuộc miền xác định của nó, và các toán tử được thiết kế cẩn thận để bảo toàn các ràng buộc này (không có vấn đề như vậy trong biểu diễn nhị phân, nhưng thiết kế của các toán tử này khá đơn giản; ta không thấy điều đó là bất lợi; mặt khác, nó lại cung cấp các lợi ích khác được trình bày dưới đây).

Sự chính xác của cách tiếp cận như thế chỉ tùy thuộc máy tính, nhưng nói chung là tốt hơn nhiều so với biểu diễn nhị phân. Dương nhiên, ta luôn có thể tăng độ chính xác của biểu diễn nhị phân khi thêm các bit, nhưng điều này làm thuật giải chậm một cách đáng kể, như đã thảo luận trong chương trước.

Thêm nữa, biểu diễn thực có khả năng biểu diễn một miền rất rộng (hoặc các trường hợp miền xác định không biết trước cụ thể). Mặt khác, trong biểu diễn nhị phân, độ chính xác sẽ giảm khi tăng kích thước miền, do chiều dài nhị phân cố định cho trước. Hơn nữa, với biểu diễn thực, việc thiết kế các công cụ đặc biệt để xử lý các ràng buộc không tầm thường sẽ dễ hơn.

5.2.2. Các toán tử chuyên biệt hóa

Các toán tử ta sẽ sử dụng rất khác các toán tử cổ điển, vì chúng làm việc trong một không gian khác (có giá trị thực). Hơn nữa, một vài toán tử không đồng bộ, nghĩa là hành động của chúng phụ thuộc vào tuổi của quần thể.

NHÓM TOÁN TỬ ĐỘT BIẾN:

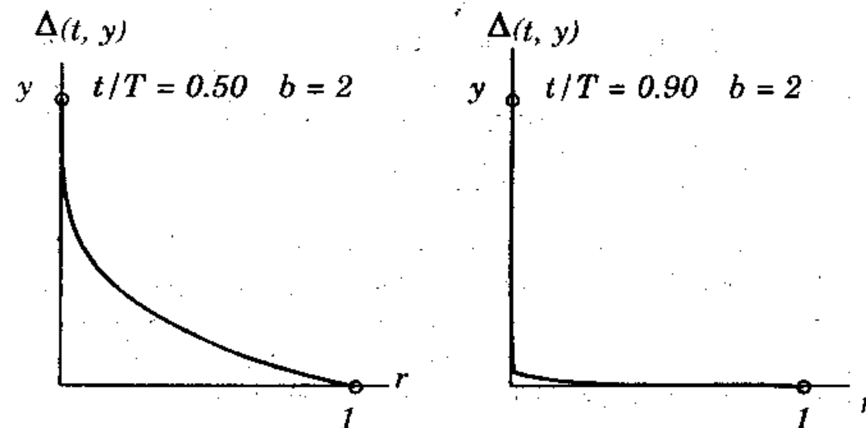
- **Đột biến đồng bộ**, được định nghĩa tương tự với định nghĩa của phiên bản cổ điển: nếu $x^t_v = \langle v_1, \ldots, v_n \rangle$ là nhiễm sắc thể, thì mỗi phần tử v_k có cơ hội trải qua tiến trình đột biến ngang nhau. Kết quả của một lần ứng dụng của toán tử này là vectơ $s^t_v = \langle v_1, \ldots, v'_k, \ldots, v_m \rangle$ và v'_k là giá trị ngẫu nhiên trong miền của tham số tương ứng.
- **Đột biến không đồng bộ** là một trong những toán tử có nhiệm vụ về tìm độ chính xác của hệ thống. Nó được định nghĩa như sau: nếu $s^t_v = \langle v_1, \ldots, v_m \rangle$ là nhiễm sắc thể và phần tử v_k được chọn đột biến này (miền của v_k là $[l_k, u_k]$), kết quả là một vectơ $s^{t+1}_v = \langle v_1, \ldots, v'_k, \ldots, v_m \rangle$ với $k \in [1, \ldots, n]$ và

$$v'_k = \begin{cases} v_k + \Delta(t, \ u_k - v_k) \ , & \text{nếu chữ số ngẫu nhiên là 0} \\ v_k - \Delta(t, \ v_k - l_k) \ , & \text{nếu chữ số ngẫu nhiên là 1} \end{cases}$$

trong đó, hàm $\Delta(t, y)$ trả về giá trị trong khoảng $[0, y]$ sao cho xác suất của $\Delta(t, y)$ gần bằng 0 sẽ tăng khi t tăng. Xác suất này buộc toán tử tìm kiếm không gian thoạt đầu là đồng bộ (khi t nhỏ), và rất cục bộ ở những giai đoạn sau. Ta sử dụng hàm sau:

$$\Delta(t, y) = y * (1 - r^{(1 - \frac{t}{T})^b}),$$

với r là số ngẫu nhiên trong khoảng [0..1], T là số thế hệ tối đa, và b là tham số hệ thống xác định mức độ không đồng bộ. Hình 5.1 biểu diễn giá trị của Δ đối với hai lần được chọn; hình này hiển thị rõ ràng cách ứng xử của toán tử.



Hình 5.1. $\Delta(t,y)$ đối với hai lần chọn.

Hơn nữa, ngoài cách áp dụng đột biến chuẩn, ta có một số cơ chế mới: đột biến không đồng bộ cũng được áp dụng cho một vectơ lời giải thay vì chỉ một phần tử duy nhất của nó, khiến cho vectơ hơi trượt trong không gian lời giải.

NHÓM TOÁN TỬ LAI TẠO

- **Lai đơn giản**, định nghĩa theo cách thông thường, nhưng chỉ với các điểm tách cho phép giữa các v', đối với nhiễm sắc thể x cho trước.
- **Lai số học**, được định nghĩa như một tổ hợp tuyến tính của hai vectơ: nếu s^t_v và s^t_w giao phối nhau, nhiễm sắc thể con có được là $s_v^{t+1} = a.s_w^t + (1-a).s_v^t$ và $s_w^{t+1} = a.s_v^t + (1-a).s_w^t$. Toán tử này có thể dùng tham số a hoặc là một hằng số (lai số học đồng bộ), hoặc là một biến mà giá trị của nó phụ thuộc tuổi của quần thể (lai số học không đồng bộ).

  Ở đây, ta có một số cơ chế mới để áp dụng toán tử này; lai số học có thể được áp dụng cho các phần tử được chọn của hai vectơ hoặc cho toàn bộ các vectơ.

5.3. THỬ NGHIỆM VÀ KẾT QUẢ

Trong phần này, chúng tôi trình bày các kết quả của chương trình tiến hóa đối với các bài toán điều khiển tối ưu. Đối với tất cả các bài toán thử nghiệm kích thước quần thể cố định là 70, và mỗi lần chạy 40000 thế hệ. Đối với mỗi trường hợp thử nghiệm, ta thực hiện ba lần chạy khác nhau và chọn kết quả tốt nhất trong ba lần chạy đó; nhưng điều quan trọng cần để ý là độ lệch chuẩn của những lần chạy đó, hầu hết đều không đáng kể. Các vectơ $<u_0,..., u_{N-1}>$ được khởi tạo ngẫu nhiên, (nhưng trong miền xác định). Các bảng 5.2, 5.3, 5.4 cho biết các giá trị tìm được cùng các kết quả trung gian tại một số thế hệ. Thí dụ, các giá trị trong cột "10,000" biểu diễn một

phần kết quả sau 10,000 thế hệ, trong khi chạy 40,000. Điều quan trọng cần lưu ý là những giá trị đó tệ hơn những giá trị nhận được trong khi chỉ chạy 10,000 thế hệ, do bản chất của một số toán tử di truyền. Trong phần kế tiếp, ta so sánh những kết quả này với những lời giải chính xác và những lời giải nhận được từ gói tính toán máy GAMS. (*)

Bảng 5.2. Kết quả chương trình tiến hóa của bài toán (5.1) - (5.2)

Trường hợp	Các thế hệ							Nhân tố
	1	100	1,000	10,000	20,000	30,000	40,000	
I	17904.4	3.87385	1.73682	1.61859	1.61817	1.61804	1.61804	10^4
II	13572.3	5.56187	1.35678	1.11451	1.09201	1.09162	1.09161	10^5
III	17024.8	2.89355	1.06954	1.00952	1.00124	1.00102	1.00100	10^7
IV	15082.1	8.74213	4.05532	3.71745	3.70811	3.70162	3.70160	10^4
V	5968.42	12.2782	2.69862	2.85524	2.87645	2.87571	2.87569	10^5
VI	17897.7	5.27447	2.09334	1.61863	1.61837	1.61805	1.61804	10^4
VII	2690258	18.6685	7.23567	1.73564	1.65413	1.61842	1.61804	10^4
VIII	123.942	72.1958	1.95783	1.00009	1.00005	1.00005	1.00005	10^4
IX	7.28165	4.32740	4.39091	4.42524	4.31021	4.31004	4.31004	10^5
X	9971391	148233	16081.0	1.48445	1.00040	1.00010	1.00010	10^4

(*) GAMS *là chương trình máy tính chuẩn dùng giải và giảng dạy các bài toán dạng này.*

Lưu ý, bài toán (5.5) - (5.7) có điều kiện ràng buộc ở giai đoạn cuối, khác với bài toán (5.1) - (5.2) ở chỗ là vectơ được khởi tạo, không ngẫu nhiên lắm, $\langle u_0, \ldots, u_{N-1}\rangle$ các số thực dương phát sinh một chuỗi x_i (xem điều kiện (5.6)) sao cho $x_0 = x_N$, với a và x_0 cho trước. Trong chương trình tiến hóa của chúng ta, ta đã phát sinh một chuỗi ngẫu nhiên $u_0, \ldots, u_{N-2}$, và đã đặt $u_{N-1} = a*x_{N-1} - x_N$. Đối với u_{N-1} âm, ta đã loại chuỗi này và lặp lại tiến trình khởi tạo (sẽ bàn chi tiết ở chương sau). Khó khăn tương tự xuất hiện trong tiến trình sinh sản. Một con (sau một số tác vụ di truyền) không còn thỏa ràng buộc $x_0 = x_N$. Trong trường hợp đó, ta thay thành phần cuối của vectơ con u bằng công thức $u_{N-1} = a*x_{N-1} - x_N$. Lần nữa, nếu u_{N-1} âm, ta sẽ không đưa đứa con đó vào quần thể mới.

Trong chương này, đây là bài toán thử nghiệm duy nhất có ràng buộc không tầm thường. Ta sẽ bàn toàn bộ vấn đề về các bài toán điều khiển tối ưu ràng buộc trong chương sau.

Bảng 5.3. Chương trình tiến hóa của bài toán thu hoạch (5.5) - (5.7)

	Các thế hệ						
N	1	100	1000	10,000	20,000	30,000	40,000
2	5.3310	5.3317	5.3317	5.3317	5.3317	5.3317	5.331738
4	12.6848	12.7172	12.7206	12.7210	12.7210	12.7210	12.721038
8	25.4601	25.6772	25.9024	25.9057	25.9057	25.9057	25.905710
10	32.1981	32.5010	32.8152	32.8209	32.8209	32.8209	32.820943
20	65.3884	68.6257	73.1167	73.2372	73.2376	73.2376	73.237668
45	167.1384	251.3241	277.3990	279.0657	279.2612	279.2676	279.271421

Bảng 5.4. Chương trình tiến hóa cho bài toán xe kéo (5.9) - (5.11)

	Các thế hệ						
N	1	100	1000	10,000	20,000	30,000	40,000
5	-3.008351	0.081197	0.119979	0.120000	0.120000	0.120000	0.120000
10	-5.668287	-0.011064	0.140195	0.142496	0.142500	0.142500	0.142500
15	-5.885241	-0.012345	0.142546	0.150338	0.150370	0.150370	0.150371
20	-7.477872	-0.126734	0.149953	0.154343	0.154375	0.154375	0.154377
25	-8.668933	-0.015673	0.143030	0.156775	0.156800	0.156800	0.156800
30	-12..257346	-0.194342	0.123045	0.158241	0.158421	0.158426	0.158426
35	-11.789546	-0.236753	0.110964	0.159307	0.159586	0.159592	0.159592
40	-10.985642	-0.235642	0.072378	0.160250	0.160466	0.160469	0.160469
45	-12.789345	-0.342671	0.072364	0.160913	0.161127	0.161152	0.161152

5.4. Chương trình tiến hóa với các phương pháp khác

Trong phần này, ta so sánh những kết quả trên với những lời giải chính xác cũng như những lời giải nhận được GAMS.

5.4.1. Bài toán tuyến tính bình phương

Lời giải chính xác của bài toán này với những giá trị của những tham số được đặc tả trong bảng 5.1, mô tả trong (5.3) và (5.4).

Để làm nổi bật hiệu quả và tính cạnh tranh của chương trình tiến hóa, những bài toán thử nghiệm tương tự đã được giải bằng GAMS. Bảng (5.5) tóm tắt các kết quả, ở đây các cột *D* chỉ phần trăm sai số tương đối.

Bảng 5.5. So sánh các lời giải của bài toán tuyến tính bình phương

Trường hợp	Lời giải chính xác	Chương trình tiến hóa		GAMS	
	Giá trị	Giá trị	D %	Giá trị	D %
I	16180.3399	16180.3928	0.000	16180.3399	0.000
II	109160.7978	109161.0138	0.000	109160.7078	0.000
III	10009990.0200	10010041.3789	0.000	10009990.0200	0.000
IV	37015.6212	37015.0426	0.000	37015.6212	0.000
V	287569.3725	287569.4357	0.000	287569.3725	0.000
VI	16180.3399	16180.4065	0.000	16180.3399	0.000
VII	16180.3399	16180.3784	0.000	16180.3399	0.000
VIII	10000.5000	10000.5000	0.000	10000.5000	0.000
IX	431004.0987	431004.4182	0.000	431004.0987	0.000
X	1000.9999	10001.0038	0.000	10000.9999	0.000

Thực ra, vì được thiết kế chuyên biệt, GAMS thực hiện bài toán tuyến tính thật hoàn hảo. Nhưng, đây lại không phải là trường hợp của bài toán thử nghiệm thứ hai.

5.4.2. Bài toán thu hoạch

Không có lời giải GAMS nào giống với lời giải chính xác. Sự khác biệt giữa các lời giải càng tăng khi *N* lớn như bảng 5.6 trình bày, và với *N* > 4, hệ thống không tìm được giá trị nào.

Có vẻ là GAMS nhạy cảm với sự không - lồi của bài toán tối ưu và với số biến.

5.4.3. Bài toán xe kéo

Cả GAMS và chương trình tiến hóa đều cho ra những kết quả tốt đối với bài toán xe kéo (bảng 5.7). Nhưng, khá thú vị nếu để ý đến mối liên hệ giữa số lần mà các thuật giải tìm kiếm khác nhau cần có để hoàn thành công việc.

N	Lời giải chính xác	GAMS		GAMS+		GT di truyền	
	giá trị	giá trị	D%	giá trị	D%	giá trị	D%
2	5.331738	4.3693	30.99	5.3316	0.00	5.3317	0.000
4	12.721038	5.9050	53.58	12.7210	0.00	12.7210	0.000
8	25.905710	*		18.8604	27.20	25.9057	0.000
10	32.820943	*		22.9416	30.10	32.8209	0.000
20	73.2337681	*		*		73.2376	0.000
45	279.275275	*		*		279.2714	0.001

Bảng 5.6. So sánh các lời giải của bài toán thu hoạch. Dấu sao (*) có nghĩa là GAMS không báo cáo được giá trị hợp lý.



N	Lời giải chính xác	GAMS		GA	
	giá trị	giá trị	D(%)	giá trị	D(%)
5	0.120000	0.120000	0.000	0.120000	0.000
10	0.142500	0.142500	0.000	0.142500	0.000
15	0.150370	0.150370	0.000	0.150370	0.000
20	0.154375	0.154375	0.000	0.154375	0.000
25	0.156800	0.156800	0.000	0.156800	0.000
30	0.158426	0.158426	0.000	0.158426	0.000
35	0.160469	0.160469	0.000	0.160469	0.000
40	0.160469	0.160469	0.000	0.160469	0.000
45	0.161152	0.161152	0.000	0.161152	0.000

Bảng 5.7. So sánh các lời giải của bài toán xe kéo.

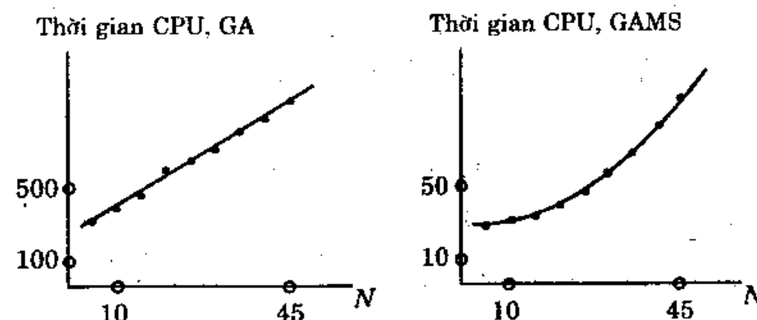
Bảng 5.8 cho biết về số lần lặp mà chương trình tiến hóa cần để đạt được lời giải chính xác (làm tròn đến 6 số lẻ), thời gian cần thiết cho việc đó, và thời gian tổng cộng cho cả 40,000 lần lặp (Nếu không biết lời giải chính xác ta không thể xác định độ chính xác của lời giải hiện hành). Cũng vậy, thời gian của GAMS là được cho trước.

Rõ ràng là chương trình tiến hóa chậm hơn GAMS nhiều: có sự khác biệt về thời gian CPU. Tuy vậy, ta không so sánh thời gian cần thiết cho cả hai hệ thống hoàn thành việc tính toán, mà so sánh tỉ lệ gia tăng thời gian - là hàm theo kích thước bài toán. Hình 5.2 biểu diễn tỉ lệ gia tăng thời gian cần thiết để nhận được kết quả của chương trình tiến hóa và GAMS.



N	Số lần lặp cần có	Thời gian (CPU) cần có (sec)	Thời gian cần cho 40,000 lần lặp (CPU sec)	Thời gian cần cho GAMS (CPU, sec)
5	6234	65.4	328.9	31.5
10	10231	109.7	400.9	33.1
15	19256	230.8	459.8	35.6
20	19993	257.8	590.8	41.1
25	18804	301.3	640.4	47.7
30	22976	389.5	701.9	58.2
35	23768	413.6	779.5	68.0
40	25634	467.8	850.7	81.3
45	28756	615.9	935.3	95.9

Bảng 5.8 Thời gian thực hiện của chương trình tiến hóa GAMS đối với bài toán xe kéo (5.9), (5.11): số lần lặp cần có để đạt kết quả chính xác đến 6 số lẻ, thời gian cần thiết cho số lần lặp đó, thời gian cần thiết cho cả 40,000 lần lặp.



Hình 5.2. Thời gian là hàm của kích thước bài toán (*N*).

5.4.4. Tầm quan trọng của đột biến không đồng bộ

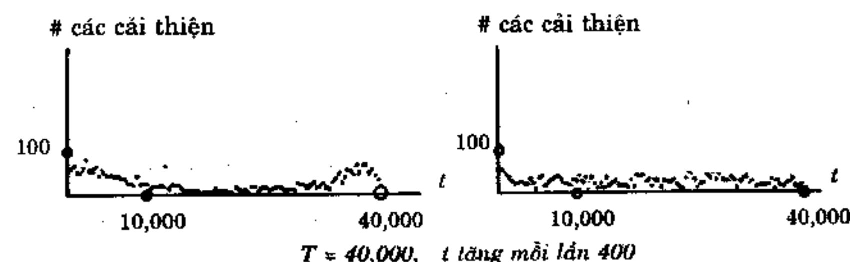
So sánh những kết quả này với những lời giải chính xác cũng như những lời giải nhận được từ GA khác sẽ thấy thú vị là chúng hoàn toàn giống nhau, nhưng không có đột biến không đồng bộ ở đó. Bảng 5.9 tóm tắt kết quả; cột *D* cho thấy sai số tương đối theo phần trăm.

Thuật giải di truyền sử dụng đột biến không đồng bộ rõ ràng thực hiện tốt hơn về độ sự chính xác của lời giải tối ưu tìm được; trong khi GA cải tiến hiếm khi đạt được sai số vượt quá vài phần ngàn, còn GA truyền thống hầu như khó vượt được một phần trăm. Hơn nữa, nó cũng đã hội tụ nhanh về lời giải.

Trường hợp	Lời giải chính xác	GA với đột biến không đồng bộ		GA không đột biến không đồng bộ	
	giá trị	giá trị	D(%)	giá trị	D(%)
I	16180.3399	16180.3939	0.000	16234.3233	0.334
II	109160.7978	109163.0278	0.000	113807.2444	4.257
III	10009990.0200	10010391.3989	0.001	10128951.4515	1.188
IV	37015.6212	37015.0806	0.001	37035.5652	0.954
V	287569.3725	287569.7389	0.000	298214.4587	3.702
VI	16180.3399	16180.6166	0.002	16238.2135	0.358
VII	16180.3399	16188.2394	0.048	17278.8502	5.786
VIII	10000.5000	10000.5000	0.000	10000.5000	0.000
IX	431004.0987	431004.4092	0.000	431610.9771	0.141
X	10000.9999	10001.0045	0.000	10439.2695	4.380

Bảng 5.9. So sánh những lời giải của bài toán điều khiển động tuyến tính bình phương

Xem hình 5.3 để có một minh họa về hiệu quả của đột biến không đồng bộ trong tiến trình tiến hóa; đột biến mới làm tăng số cải thiện thấy được trong quần thể vào cuối đời sống của quần thể rất nhiều. Hơn nữa, một số nhỏ những cải thiện trước thời gian đó, cùng với một hội tụ thực sự nhanh hơn, rõ ràng đã cho thấy một tìm kiếm tổng thể tốt hơn.



Hình 5.3. Số cải thiện trong trường hợp 1 của bài toán điều khiển động tuyến tính bình phương.

5.5. KẾT LUẬN

Trong chương này, ta đã nghiên cứu một toán tử mới, *đột biến không đồng bộ*, để cải thiện các khả năng tìm chính xác cục bộ của một GA. Các thử nghiệm đã thành công trên những bài toán điều khiển tối ưu thời gian rời rạc. Đặc biệt, các kết quả rất đáng khích lệ vì các lời giải số đã đạt rất gần các lời giải giải tích một cách đáng hài lòng. Thêm nữa, thời gian máy tính toán là chấp nhận được (đối với 40,000 thế hệ, chỉ cần vài phút thời gian).

Các kết quả số đã được so sánh với những kết quả thu được với GAMS. Trong khi chương trình tiến hóa cho ta những kết quả tốt đáng kể với các lời giải giải tích cho tất cả các bài toán thử nghiệm, GAMS thất bại đối với một số các bài toán thử nghiệm. Chương trình tiến hóa đã cài đặt thể hiện một số tính chất không phải luôn luôn hiện diện trong các hệ thống (dựa trên đạo hàm) khác:

- Hàm tối ưu hóa của chương trình tiến hóa không cần liên tục.
- Một số chương trình tối ưu hóa phải thực hiện trọn gói: người sử dụng phải đợi cho đến khi chương trình hoàn tất. Đôi khi không thể có được một phần kết quả (hoặc xấp xỉ ở một số giai đoạn đầu). Các chương trình tiến hóa cung cấp thêm một số tính linh hoạt cho người sử dụng, vì người sử dụng có thể điều khiển "tình trạng tìm kiếm" trong thời gian chạy và ra những quyết định thích hợp. Đặc biệt, người sử dụng có thể đặc tả thời gian máy mà mình có thể chấp nhận (thời gian lâu hơn cung cấp câu trả lời chính xác hơn).
- Độ phức tạp của tính toán trong những chương trình tiến hóa tăng theo tỉ lệ tuyến tính; hầu hết những phương pháp tìm kiếm khác rất nhạy cảm với tỷ lệ này. Ta cũng có thể dễ dàng cải thiện hiệu quả của hệ thống bằng những cài đặt song song; đối với những phương pháp tối ưu hóa khác thường khó thực hiện như vậy.

Gần đây có nhiều phát triển thú vị trong lãnh vực thuật giải tiến hóa để nâng cao khả năng chính xác của chúng. Những phương pháp này là thuật giải mã hóa Delta, Mã Hóa Tham Số Động, chiến lược ARGOT, các chiến lược IRM, mở rộng của lập trình tiến hóa, tiến hóa thô, và các thuật giải di truyền theo quãng.

Có một số hoạt động khác (ít nhiều trực tiếp hơn) nhằm vào các khả năng chính xác. Những hoạt động này, gồm nghiên cứu của Arabas và những người khác, giới thiệu các phép lai trung gian bổ sung và đồng nhất của các chiến lược tiến hóa (xem chương 7). Hinterding cũng đã thử nghiệm với đột biến của các gen (tương ứng với các biến) thay cho đột biến bit. Từ đó, các tác giả phân tích ý nghĩa của mã hóa (Gray đối chiếu với Nhị phân), kết cấu thô, và tần suất của các đột biến gen như thế.

Cũng có những kết quả thú vị do Srinivas và Patnaik báo cáo, họ đã thử nghiệm với các xác suất đột biến và lai bổ sung để duy trì sự đa dạng của quần thể (và duy trì khả năng hội tụ của thuật giải). Trong cách tiếp cận này, các xác suất của những toán tử này thay đổi theo các giá trị thích nghi của những lời giải: những lời giải "tốt" được bảo vệ còn những lời giải "tồi" thì bị hủy. Nói chính xác hơn:

$$p_c = \begin{cases} k_1.(f_{max} - f') \;/\; (f_{max} - \overline{f}), & \text{nếu } f' \le \overline{f} \\ k_3, & \text{ngược lại} \end{cases}$$

và

$$p_c = \begin{cases} k_2.(f_{max} - f') \;/\; (f_{max} - \overline{f}), & \text{nếu } f' \le \overline{f} \\ k_4, & \text{ngược lại} \end{cases}$$

Trong đó, k_1 và k_2 là những hằng dương (không lớn hơn 1), f_{max} và $\overline{f}$ lần lượt cho biết các giá trị cực đại và trung bình của hàm thích nghi f trong quần thể hiện tại, f là giá trị của hàm thích nghi đối với một lời giải cho trước, còn f' - giá trị lớn hơn (đối với hai lời giải được chọn để lai tạo). Chú ý rằng (1) giá trị của $f_{max} - \overline{f}$ là cần thiết trong công thức trên; nó cũng quan trọng trong việc đo độ hội tụ của thuật giải; (2) p_c và p_m bằng 0 đối với lời giải có thích nghi cực đại; (3) $p_c = k_1$ và $p_m = k_2$ đối với lời giải có $f = \overline{f}$ và (4) $p_c = k_3$ và $p_m = k_4$ đối với những lời giải dưới trung bình.

Chương 6

XỬ LÝ RÀNG BUỘC

Bài toán qui hoạch phi tuyến tổng quát – NLP (nonlinear programming problem) là tìm x để hàm:

$$f(x),\ \ x = (x_1, ..., x_q) \in R^q$$

đạt giá trị tối ưu thỏa p ≥ 0 phương trình:

$$g_j(x) = 0,\ j = 0,\ ...,\ p,$$

và $m - p \geq 0$ bất phương trình:

$$h_j(x) \leq 0,\ j = p + 1,\ ...,\ m$$

Hiện chưa có phương pháp nào giúp xác định cực đại (hay cực tiểu) toàn cục cho bài toán NLP tổng quát. Chỉ khi hàm mục tiêu f và các ràng buộc c_j (g_j và h_j), thỏa một số tính chất nào đó, thì đôi khi mới tìm được tối ưu toàn cục. Nhiều thuật giải được phát triển cho những bài toán không ràng buộc (phương pháp tìm kiếm trực tiếp, phương pháp gradient) và những bài toán có ràng buộc (những thuật giải này thường được xếp vào một trong hai lớp: lớp phương pháp gián tiếp và lớp trực tiếp). Phương pháp gián tiếp giải bài toán NLP bằng cách trích ra một (số) bài toán tuyến tính từ bài toán gốc, trong khi phương pháp trực tiếp cố gắng xác định trực tiếp những điểm tìm kiếm. Để thực hiện, bài toán gốc được biến đổi thành bài toán không ràng buộc, để có thể áp dụng những phương pháp gradient, hay phương pháp gradient cải tiến. Mặc dầu những nghiên cứu về bài toán NLP trong những năm gần đây đã có những kết quả tích cực, nhưng công bình mà nói, chưa có thuật giải nào giải hiệu quả bài toán NLP tổng quát.

Có nhiều bài toán khác có liên quan đến các kỹ thuật tối ưu hóa truyền thống. Thí dụ, hầu hết các phương pháp được đề nghị là các phương pháp tầm cục bộ, chúng phụ thuộc vào sự tồn tại các đạo hàm và chúng không đủ mạnh trong những không gian không liên tục, miền xác định lớn, hay không gian tìm kiếm bị nhiễu.

Trong chương này, ta sẽ xây dựng các chương trình tiến hoá giải bài toán NLP này. Đầu tiên, ta sẽ giải bài toán NLP có không gian tìm kiếm lồi. Sau đó, ta sẽ mở rộng cho bài toán NLP tổng quát.

6.1. Bài toán qui hoạch phi tuyến trên không gian lồi

Nhiều nhà nghiên cứu đã thử sử dụng GA với biểu diễn thực vào bài toán tối ưu số. Tuy nhiên, các bài toán tối ưu này chỉ bị ràng buộc theo miền $D \subseteq R^q$, với $D = \prod_{k=1}^{q} < l_k, r_k >$, nghĩa là mỗi biến x_k bị giới hạn trong một khoảng cho trước $<l_k, r_k>$ ($1 \le k \le q$). Nhưng bài toán tối ưu tổng quát thường gặp trong thực tế hầu hết lại là bài toán tối ưu có ràng buộc.

Trong bài toán tối ưu có ràng buộc, dạng hình học của tập lời giải trong R^q có lẽ là đặc trưng quan trọng nhất của bài toán. Hiện chỉ có tập lồi là có phát triển lý thuyết đáng kể nhất.

Trong phần này, ta quan tâm đến bài toán tối ưu hàm $f(x_1,...,x_q) \in R$,

với $(x_1,...,x_q) \in D \subseteq R^q$ và D là một tập *lồi*.

Miền D được xác định từ các khoảng giá trị ($l_k \le x_k \le r_k$ với $k= 1,...q$) và từ tập các ràng buộc C. Do tính lồi của tập D mà với mỗi điểm trong không gian tìm kiếm $(x_1,...,x_q) \in D$, tồn tại một đoạn $<left(k), right(k)>$ của biến x_k ($1 \le k \le q$), trong đó các biến x_i

($i=1,...,k-1, k+1,...,q$) khác vẫn cố định. Nói cách khác, điểm $(x_1,...,x_k,...,x_q) \in D$ cho trước:

$$y \in <left(k), right(k)> \Leftrightarrow (x_1,...x_{k-1},y,x_{k+1},...,x_q) \in D.$$

trong đó, tất cả x_i ($i=1,...,k-1, k+1,...,q$) vẫn là hằng số. Ta cũng giả định khoảng $<left(k), right(k)>$ có thể tính được bằng máy.

Thí dụ, nếu $D \subseteq R^2$ được định nghĩa là:

$-3 \le x_1 \le 3$,

$0 \le x_2 \le 8$,

$x_1^2 \le x_2 \le x_1 + 4$,

thì đối với điểm $(2, 5) \in D$ cho trước:

$left(1) = 1, right(1) = \sqrt{5}$

$left(2) = 4, right(2) = 6$

Điều này có nghĩa là thành phần thứ nhất của vectơ (2, 5) có thể thay đổi từ 1 đến $\sqrt{5}$ (trong khi $x_2 = 5$ vẫn không đổi) và thành phần thứ hai của vectơ này có thể thay đổi từ 4 đến 6 (trong khi $x_1 = 2$ vẫn giữa nguyên).

Dĩ nhiên, nếu tập các ràng buộc C rỗng, thì không gian tìm kiếm $D = \prod_{k=1}^{q} < l_k, r_k >$ là lồi; hơn nữa $left(k) = l_k$; $right(k) = r_k$ với $k = 1, ..., q$.

Tính chất trên là cơ sở để ta xây dựng các toán tử đột biến; nếu biến x_k bị đột biến, khoảng đột biến phải là $<left(k), right(k)>$; và như thế, kết quả luôn thỏa ràng buộc.

Một tính chất khác của không gian tìm kiếm lồi là bảo đảm rằng, với hai điểm x_1, x_2 bất kỳ trong không gian lời giải D, tổ hợp tuyến tính $ax_1+(1-a)x_2$, với $a \in [0, 1]$ cũng là điểm trong D. Tính chất này quan trọng khi ta xây dựng các phép lai số học.

Ta sẽ xét một lớp bài toán tối ưu được xác định trên một miền lồi; những bài toán này được hình thức hóa như sau:

Tối ưu hàm $f(x_1, x_2, ..., x_q)$ với tập ràng buộc tuyến tính:

1. Ràng buộc về miền $l_i \le x_i \le u_i$. Ta viết $\boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}$, trong đó, $\boldsymbol{l} = <l_1,...,l_q>$, $\boldsymbol{u} = <u_1,...,u_q>$, $\boldsymbol{x} = <x_1,...,x_q>$.
2. Ràng buộc đẳng thức $A\boldsymbol{x} = \boldsymbol{b}$, trong đó, $\boldsymbol{x} = <x_1,...,x_q>$, $A = (a_{ij})$, $\boldsymbol{b} = <b_1,...,b_p>$, $1 \le i \le p$ và $1 \le j \le q$ (p là số phương trình).
3. Ràng buộc bất đẳng thức $C\boldsymbol{x} <= \boldsymbol{d}$, trong đó, $C = (c_{ij})$, $\boldsymbol{x} = <x_1,...,x_m>$, $\boldsymbol{d} = <d_1,...,d_m>$, $1 \le i \le m$ và $1 \le j \le m$ (m là số bất đẳng thức).

Cách hình thức hóa như thế thường đủ để xử lý một lớp lớn các bài toán tối ưu với các ràng buộc tuyến tính của một hàm mục tiêu bất kỳ. Thí dụ bài toán vận tải phi tuyến, là một trong những bài toán thuộc lớp này.

Hệ thống GENOCOP, được phát triển bởi Michalewicz, cài đặt cách xử lý ràng buộc vừa tổng quát vừa độc lập bài toán. Ý tưởng chính là (1) loại trừ các đẳng thức có trong tập ràng buộc, và (2) thiết kế kỹ lưỡng các toán tử di truyền đặc biệt, đảm bảo giữ được tất cả các nhiễm sắc thể trong không gian lời giải ràng buộc.

Trong một số kỹ thuật tối ưu hóa như qui hoạch tuyến tính, các ràng buộc về đẳng thức rất dễ cho việc giải bài toán tối ưu, do ta biết rằng, nếu lời giải tối ưu tồn tại, nó sẽ thuộc bề mặt của tập lồi. Vì thế, các bất đẳng thức sẽ được chuyển thành các đẳng thức bằng

cách thêm vào các biến tạm, và phương pháp lời giải tiến hành bằng cách di chuyển từ đỉnh này đến đỉnh khác trên bề mặt.

Ngược lại, đối với phương pháp phát sinh các lời giải ngẫu nhiên như GA, thì các ràng buộc đẳng thức như thế lại thật phiền toái. Chúng được loại trừ ngay từ đầu với cùng số biến bài toán; hành động này cũng xóa đi một phần không gian cần tìm kiếm. Các ràng buộc còn lại, dưới dạng các bất đẳng thức tuyến tính, tạo thành một tập lồi phải tìm cho lời giải. Tính lồi của không gian tìm kiếm bảo đảm rằng tổ hợp tuyến tính các lời giải cũng sẽ là một lời giải mà không cần kiểm tra ràng buộc – đây là đặc trưng được dùng chính trong phương pháp này. Các bất đẳng thức có thể được dùng để phát sinh biên của một biến bất kỳ cho trước: những biên này là động vì chúng phụ thuộc vào các giá trị của những biến khác và có thể được tính toán tự động một cách hiệu quả.

Giả sử tập ràng buộc đẳng thức được biểu diễn dưới dạng ma trận:

$$A\boldsymbol{x} = \boldsymbol{b}$$

Ta giả định rằng có p phương trình độc lập, nghĩa là có p biến $x_{i1}, x_{i2},...x_{ip}$ ($\{i_1,..., i_p\} \subseteq \{1, 2,..., q\}$) có thể được biểu diễn theo các biến khác. Do đó, chúng có thể bị loại như sau.

Ta tách dọc mảng A thành hai mảng A_1 và A_2 sao cho cột thứ j của ma trận A thuộc A_1, nếu và chỉ nếu $j \in \{i_1,...,i_p\}$. Như vậy, A_1^{-1} tồn tại. Tương tự, ta tách ma trận C và các vectơ $\boldsymbol{x}$, $\boldsymbol{l}$, $\boldsymbol{u}$ (nghĩa là $\boldsymbol{x}^1 = <x_{i1},...,x_{ip}>$, $\boldsymbol{l}_1 = <l_{i1},...,l_{ip}>$ và $\boldsymbol{u}_1 = <u_{i1},...,u_{ip}>$) . Rồi thì ta có:

$$A_1\boldsymbol{x}^1 + A_2\boldsymbol{x}^2 = \boldsymbol{b}$$

và dễ dàng thấy rằng:

$x^1 = A_1^{-1}b - A_1^{-1}A_2x^2$

Như thế, ta có thể bỏ các biến x_{i_1}, ..., x_{i_p} và thay chúng bằng tổ hợp tuyến tính của các biến còn lại. Nhưng mỗi biến x_{i_j} (j= 1,2, ..., p) bị ràng buộc thêm bởi một ràng buộc miền $l_{i_j} \le x_{i_j} \le u_{i_j}$. Khi loại bỏ tất cả các biến x_{i_j}, sẽ cho ta một tập các bất đẳng thức mới,

$$l_1 \le x^1 = A_1^{-1}b - A_1^{-1}A_2x^2 \le u_1$$

tập này được thêm vào tập các bất đẳng thức gốc.

Tập gốc $Cx \le d$,

có thể được biểu diễn là

$$C_1x^1 + C_2x^2 \le d$$

và có thể biến đổi thành

$$C_1(A_1^{-1}b - A_1^{-1}A_2x^2) + C_2x^2 \le d$$

Như vậy, sau khi bỏ p biến x_{i_1}, ..., x_{i_p}, tập các ràng buộc cuối cùng chỉ gồm có các bất đẳng thức sau đây:

1. Các ràng buộc miền: $l_2 \le x^2 \le u_2$,
2. Các bất đẳng thức mới: $l_1 \le x^1 = A_1^{-1}b - A_1^{-1}A_2x^2 \le u_1$,
4. Các bất đẳng thức gốc (sau khi bỏ x^1 biến):

$$(C_2 - C_1A_1^{-1}A_2)x_2 \le d - C_1A_1^{-1}b$$

6.1.1. Một thí dụ minh họa

Giả sử ta muốn tối ưu hóa hàm 6 biến:

$f(x_1,x_2,x_3,x_4,x_5,x_6)$

với

$2x_1+x_2+x_3 = 6$

$x_3+x_5-3x_6 = 10$

$x_1+4x_4 = 3$

$x_2+x_5 \le 120$

$-40 \le x_1 \le 20, \quad 50 \le x_2 <= 75$

$0 \le x_3 \le 10, \quad 5 \le x_4 \le 15,$

$0 \le x_5 \le 20, \quad -5 <= x_6 \le 5$

Có 3 phương trình và 6 biến, như vậy sẽ có ba biến tự do. Ta sẽ dùng ba phương trình độc lập và biểu diễn các biến qua ba biến còn lại:

$x_1 = 3-4x_4,$

$x_2 = -10+8x_4+x_5-3x_6,$

$x_3 = 10-x_5+ 3x_6.$

Như vậy, ta đã biến bài toán gốc thành bài toán tối ưu hóa một hàm ba biến x_4, x_5, x_6.

$g(x_4, x_5, x_6) = f((3-4x_4), (-10+8x_4+x_5-3x_6), (10-x_5+ 3x_6))$

$-10 + 8x_4 + 2x_5 - 3x_6 \le 120$ (**vì** $x_5+x_2 \le 120$)

$-40 \le 3 - x_4 \le 20$, (vì $-40 \le x_1 \le 20$),

$50 \le -10 + 8x_4 + x_5 - 3x_6 \le 75$ (vì $50 \le x_2 \le 75$)

$0 \le 10 - x_5 + 3x_6 \le 10$ (vì $0 \le x_3 \le 10$)

$5 \le x_4 \le 15$, $0 \le x_5 \le 20$ và $-5 \le x_6 \le 5$.

Đến đây có thể lại giảm thêm; chẳng hạn các bất đẳng thức thứ hai và thứ năm có thể được biểu diễn bằng một bất đẳng thức duy nhất:

$5 \le x_4 \le 10.75$

Biến đổi này hoàn thành bước đầu tiên của phương pháp: loại bớt các bất đẳng thức. Đương nhiên, không gian tìm kiếm có được là không gian lồi. Như đã trình bày trước, do tính chất lồi của không gian tìm kiếm, điều tiếp theo là với mỗi điểm có được (x_1,x_2,x_3) sẽ có được một khoảng *<left(k), right(k)>* tương ứng của một biến x_k ($1\le k \le 3$), trong đó, hai biến còn lại giữ cố định. Thí dụ, đối với không gian xác định như trên, và đối với một điểm cho trước (x_4, x_5,x_6) là (10, 8, 2),

left(1)= 7.25, *right*(1) =10.375,

left(2)= 6, *right*(2) = 11,

left(3)= 1, *right*(3) = 2.666,

(*left(1)* và *right(1)* là các khoảng của thành phần đầu tiên của vectơ (10, 8, 2), nghĩa là của biến x_4, v.v...). Có nghĩa là thành phần đầu tiên của vectơ (10, 8, 2) có thể biến thiên từ 7.25 đến 10.375 (trong khi $x_5 = 8$ và $x_6 = 2$ vẫn không đổi), thành phần thứ hai của vectơ có thể biến thiên từ 6 đến 11 (trong khi $x_4 = 10$ và $x_6 = 2$ vẫn không đổi), và thành phần thứ ba của vectơ có thể biến thiên từ 1 đến 2.666 (trong khi $x_4 = 10$ và $x_5 = 8$ vẫn không đổi).

Hệ thống GENOCOP sẽ định vị một lời giải khởi tạo (khả thi) trong vùng cho phép. Nếu một số lần thử được xác định trước không thành công, hệ thống sẽ nhắc người sử dụng về điểm khởi tạo. Quần thể ban đầu gồm những bản sao tương tự của điểm ban đầu đó (dù là được người sử dụng phát sinh hay cung cấp).

Tiếp theo, ta xem xét các phép toán di truyền của GENOCOP

6.1.2. Toán tử

Trong phần này ta bàn về 6 phép toán di truyền với biểu diễn thực. Ba phép toán đầu tiên là toán tử một ngôi (loại đột biến), ba phép toán kia là toán tử 2 ngôi (các loại lai khác nhau).

ĐỘT BIẾN ĐỒNG DẠNG

Toán tử này cần một cá thể $\boldsymbol{x}$ và tạo ra một con duy nhất $\boldsymbol{x}'$. Toán tử này chọn một thành phần ngẫu nhiên $k \in (1,..., q)$ của vectơ $\boldsymbol{x} = (x_1,...,x_k,..., x_q)$, và sản sinh ra $\boldsymbol{x}' = (x_1,...,x'_k, ..., x_q)$, trong đó, x'_k là giá trị ngẫu nhiên (phân bố xác suất đều) trong khoảng *<left(k), right(k)>*.

Toán tử này đóng vai trò quan trọng trong những giai đoạn đầu của quá trình tiến hóa khi các lời giải được phép di chuyển tự do trong không gian tìm kiếm. Đặc biệt, toán tử cần thiết trong trường hợp quần thể ban đầu gồm nhiều bản sao của cùng một điểm. Tình trạng như thế có thể thường xuyên xảy ra trong những bài toán tối ưu có ràng buộc mà người sử dụng đặc tả điểm khởi đầu của tiến trình. Hơn nữa, điểm khởi đầu duy nhất này (không kể những khuyết điểm của nó) có một thuận lợi to lớn: nó cho phép phát triển một tiến trình lặp mà lần lặp kế tiếp bắt đầu tại điểm tốt nhất của

lần lặp trước. Chính kỹ thuật này đã được dùng trong việc phát triển một hệ thống xử lý các ràng buộc phi tuyến trong những không gian không nhất thiết là lồi sau này.

Cũng vậy, trong những giai đoạn sau của quá trình tiến hóa toán tử này cho phép thoát khỏi tối ưu cục bộ để tìm một điểm tốt hơn.

ĐỘT BIẾN BIÊN

Toán tử này cũng cần một cá thể cha x và tạo ra một con duy nhất x'. Toán tử này là một biến thể của đột biến đồng dạng với x'_k là *left(k)* hoặc *right(k)*, với cùng xác suất.

Toán tử được xây dựng cho các bài toán tối ưu mà lời giải tối ưu nằm trên hoặc gần biên của không gian tìm kiếm khả thi. Do đó, nếu tập ràng buộc *C* rỗng và biên thật lớn, thì toán tử này gây phiền toái. Nhưng nó lại có ích vô cùng khi có các ràng buộc. Thí dụ đơn giản này chứng minh sự tiện dụng của toán tử này; Đó là bài toán qui hoạch tuyến tính; trong trường hợp như thế ta biết lời giải toàn cục nằm ngay trên biên của không gian tìm kiếm.

Thí dụ 6.1

Xét trường hợp thử nghiệm sau:

Max $f(x_1,x_2) = 4x_1+3x_2$

$2x_1+3x_2 \le 6$,

$-3x1 +2x2 \le 3$,

$2x_1 + x_2 \le 4$

$0 \le x_i \le 2,\ i=1,2.$

Cực đại toàn cục lý thuyết là $(x_1,x_2) = (1.5,\ 1.0)$ và $f(1.5,\ 1.0) = 9.0$ để xác định sự tiện dụng của toán tử biên trong việc tối ưu bài toán trên 10 thử nghiệm đã được chạy và tất cả các toán tử đều hoạt động cùng với 10 thử nghiệm khác không có đột biến biên. Hệ thống có đột biến biên tìm được tối ưu toàn cục một cách dễ dàng trong tất cả các lần chạy, trung bình trong khoảng 32 thế hệ, trong khi không có toán tử này thì thậm chí trong 100 thế hệ điểm *tốt nhất* tìm được (trong 10 lần chạy) là $x = (1.501,\ 0.997)$ và $f(x) = 8.996$ (điểm xấu nhất là $x = (1.576,\ 0.847)$ và $f(x) = 8.803$).

ĐỘT BIẾN KHÔNG ĐỒNG DẠNG

Đây là toán tử (một ngôi) chịu trách nhiệm về khả năng nâng cao độ chính xác của hệ thống. Nó được định nghĩa như sau. Đối với cá thể cha x, nếu phần tử x_k được chọn cho đột biến này, kết quả là $x' = <x_1,\ldots,x'_k,\ldots,x_q>$, trong đó:

$$x'_k = \begin{cases} x_k + \Delta(t,\quad right(k) - x_k) & \text{nếu chữ số nhị phân ngẫu nhiên là 0} \\ x_k - \quad \Delta(t,\quad x_k - left(k)) & \text{nếu chữ số nhị phân ngẫu nhiên là 1} \end{cases}$$

Hàm $\Delta(t, y)$ trả về một giá trị trong khoảng $[0, y]$ sao cho xác suất của $\Delta(t, y)$ gần bằng 0 tăng khi t tăng (t là số thế hệ). Thuộc tính này khiến toán tử ban đầu sẽ tìm trong không gian đồng dạng (khi t nhỏ), và rất cục bộ ở những giai đoạn sau. Ta dùng hàm sau đây:

$$\Delta(t,y) = y.r.(1-\frac{t}{T})^b$$

trong đó, r là một số ngẫu nhiên trong khoảng $[0..1]$, T là số thế hệ tối đa, còn b là tham số hệ thống, xác định mức độ không đồng dạng.

LAI SỐ HỌC

Toán tử nhị phân này được định nghĩa là tổ hợp tuyến tính của hai vectơ: x_1 và x_2, các con sinh ra sẽ là $x'_1 = ax_1 + (1-a)x_2$ và $x'_2 = ax_2 + (1-a)x_1$. Toán tử này dùng giá trị ngẫu nhiên $a \in [0..1]$, vì nó luôn bảo đảm $(x'_1, x'_2 \in D)$. Lai như thế được gọi là *lai trung bình có bảo đảm ràng buộc* (khi a = 1/2); *lai tức khắc*; *lai tuyến tính*; hay *lai số học*.

Sự quan trọng của phép lai số học được minh họa trong thí dụ sau.

Thí dụ 6.2

Xét bài toán sau:

$$\text{Min } f(x_1,x_2,x_3,x_4,x_5) = -5\sin(x_1)\sin(x_2)\sin(x_3)\sin(x_4)\sin(x_5)$$
$$-\sin(5x_1)\sin(5x_2)\sin(5x_3)\sin(5x_4)\sin(5x_5),$$

$0 \le x_i \le \pi$, với $1 \le i \le 5$.

Lời giải toàn cục là $(x_1,x_2,x_3,x_4,x_5) = (\pi/2, \pi/2, \pi/2, \pi/2, \pi/2)$, và $f(\pi/2, \pi/2, \pi/2, \pi/2, \pi/2) = -6$.

Hình như hệ thống không – lai số học hội tụ chậm hơn. Sau 50 thế hệ, giá trị trung bình của điểm tốt nhất (sau 10 lần chạy) là –5.9814, và giá trị trung bình của điểm tốt nhất sau 100 thế hệ là–5.9966. Trong khi giá trị trung bình đối với hệ lai số học theo thứ tự là –5.9930 và –5.9996.

Thêm nữa, một điểm thú vị là hệ thống có lai số học ổn định hơn, với độ lệch chuẩn của những lời giải tốt nhất (có được trong 10 lần chạy) thấp hơn nhiều.

LAI ĐƠN GIẢN

Toán tử nhị phân này được định nghĩa như sau: nếu kết hợp x1 = $(x_1, \ldots, x_q)$ và x2 = $(y_1,\ldots,y_q)$ thì sinh được các con là $x'1 = (x_1,\ldots,x_k,y_{k+1},\ldots,y_q)$ và $x'_2 = (y_1,\ldots,y_k, x_{k+1},\ldots,x_q)$. Một toán tử như thế có thể sinh các con ngoài miền D. Để tránh điều này, ta sử dụng thuộc tính của không gian lồi trong đó tồn tại $a \in [0..1]$ sao cho:

$$x'_1 = <x_1,\ldots,x_k,y_{k+1}{*}\ a + x_{k+1}{*}(1-a),\ldots,y_q{*}a + x_q(1-a)>$$

và

$$x'_2 = <y_1,\ldots,y_k,x_{k+1}{*}\ a + y_{k+1}{*}(1-a),\ldots,xq{*}a + y_q(1-a)>$$

là thỏa mãn ràng buộc.

Vấn đề còn lại cần giải quyết là cách tìm số a lớn nhất để sự trao đổi thông tin là nhiều nhất. Phương pháp đơn giản nhất là bắt đầu với $a = 1$ và; nếu ít nhất có một con không thuộc D, thì giảm số a bằng vài hằng số $1/\rho$. Sau ρ lần thử $a = 0$ và cả hai con đều thuộc D vì chúng đồng nhất với cha mẹ của chúng. Điều cần thiết của việc giảm tối đa như thế nói chung nhỏ và giảm nhanh qua suốt đời sống quần thể.

Dường như phẩm chất của phép lai đơn giản cũng giống như của lai số học. Các kết quả thực nghiệm cho thấy hàm không – lai đơn giản thậm chí còn ít ổn định hơn hệ không – lai số học; trong trường hợp này, độ lệch chuẩn của lời giải tốt nhất trong 10 lần chạy cao hơn nhiều. Cũng thế, lời giải xấu nhất nhận được trong 100 thế hệ có giá trị –5.9547, quá xấu so với lời giải tối nhất nhận được với tất cả các toán tử (–5.9982) hay lời giải xấu nhất nhận được mà không có lai số học (–5.9919).

LAI HEURISTIC

Toán tử này là phép lai một con vì những lý do sau: (1) nó dùng các giá trị của hàm mục tiêu để quyết định hướng tìm kiếm, (2) nó chỉ tạo ra một con, và (3) cũng có thể không tạo ra con nào cả.

Toán tử này phát sinh một con duy nhất x = từ hai cá thể cha mẹ x_1 và x_2 theo luật sau đây:

$$x_3 = r^*(x_2 - x_1) + x_2,$$

trong đó, r là số ngẫu nhiên giữa 0 và 1, và x_2 không xấu hơn x_1, nghĩa là, $f(x_2) \geq f(x_1)$ đối với những bài toán cực đại hóa và $f(x_2) \leq f(x_1)$ đối những bài toán cực tiểu hóa.

Có thể toán tử này phát sinh một vectơ con không thỏa ràng buộc. Trường hợp này, một giá trị ngẫu nhiên r khác được phát sinh và một con khác được tạo. Nếu sau w lần thử mà không tìm được một lời giải mới nào thỏa các ràng buộc, toán tử này sẽ bỏ cuộc và sẽ không tạo ra con nào.

Dường như phép lai heuristic đóng góp độ chính xác cho lời giải tìm được; các trách nhiệm của nó là (1) tìm chính xác cục bộ, và (2) tìm theo hướng hứa hẹn nhất.

6.2. Tối ưu hàm phi tuyến

Trong phần này, chúng tôi trình bày một hệ thống 'lai', hệ GENOCOP II, giải bài toán qui hoạch phi tuyến. Hệ thống được xây dựng dựa trên những ý tưởng rút ra từ những cài đặt gần đây trong lãnh vực tối ưu hóa kết hợp với những kinh nghiệm thu được từ hệ GENOCOP.

Ta giả thiết rằng hàm mục tiêu $f(x)$ và tất cả các ràng buộc là những hàm liên tục khả vi bậc hai theo x. Cách tiếp cận tổng quát là biến đổi bài toán *NLP* thành một chuỗi các bài toán nhỏ dễ giải hơn. Những phương pháp này yêu cầu cách tính đạo hàm bậc hai tường minh (hay không tường minh) của hàm mục tiêu (hay hàm đã biến đổi).

Một trong các cách tiếp cận là phương pháp hàm thưởng phạt bình phương tuần tự, được vận dụng như ý tưởng chủ đạo hậu thuẫn cho hệ GENOCOP II. Phương pháp này thay bài toán NLP bằng NLP':

$$\text{Tối ưu } F(x, r) = f(x) + \frac{1}{2r}\overline{C}^T\overline{C}$$

Trong đó, $r > 0$ và $\overline{C}$ là vectơ của tất cả ràng buộc hiện hành $c_1,\ldots,c_l$.

Fiacco và McCormick đã chứng minh được rằng khi $r \to 0$, lời giải của NLP và NLP' là tương đương.

Hình 7.1 trình bày các bước chính của hệ GENOCOP II.

Thủ tục GENOCOP II

Bắt đầu

$t \leftarrow 0$

Tách tập các ràng buộc C thành,

$$C = L \cup N_e \cup N_i$$

Chọn điểm khởi đầu x_s



Tạo tập A các ràng buộc hiện hành,

$A \leftarrow N_e \cup V$

Đặt $\tau \leftarrow \tau_0$

Khi chưa thỏa (điều kiện dừng) **Làm**

$t \leftarrow t+1$

Thi hành GENOCOP với hàm:

$$F(x,\tau) = f(x) + \frac{1}{2\tau}\overline{A}^T\overline{A}$$

với tập ràng buộc tuyến tính L

và điểm khởi đầu x_s

Lưu cá thể tốt nhất x^*:

$x_s \leftarrow x^*$

Cập nhật A:

$A \leftarrow A - S \cup V,$

Giảm mức phạt τ :

$\tau \leftarrow g(\tau, t)$

Hết lặp

Kết thúc

Hình 6.1. Mã giả thủ tục GENOCOP II

Trong thuật giải GENOCOP II, trước khi vào vòng lặp cần thực hiện một số bước khởi tạo. Tham số t (dùng để đếm số lần lặp, nghĩa là số lần thi hành GENOCOP) có khởi tạo đầu bằng 0. Tập C các ràng buộc được chia thành ba tập con: ràng buộc tuyến tính L, phương trình phi tuyến N_e và bất đẳng thức phi tuyến N_i. Điểm khởi



đầu x_s (không nhất thiết thỏa ràng buộc) cho tiến trình tối ưu theo sau được chọn (hoặc người sử dụng được nhập vào). Tập các ràng buộc hiện hành A khởi đầu gồm có các thành phần N_e và tập $V \subseteq N_i$ của các ràng buộc bị vi phạm từ N_i. Một ràng buộc $c_j \in N_i$ bị vi phạm tại điểm x nếu và chỉ nếu $c_j(x) > \delta$ ($j = t+1, ..., m$), ở đây δ là tham số hệ thống. Sau cùng, hệ số thưởng phạt τ được khởi tạo giá trị đầu τ_0 (tham số hệ thống).

Trong thân vòng lặp, ta áp dụng GENOCOP để tối ưu hóa hàm:

$$F(x, \tau) = f(x) + \frac{1}{2\tau}\overline{A}^T\overline{A}$$

với các ràng buộc tuyến tính L.

Chú ý rằng quần thể khởi tạo của GENOCOP gồm các bản sao giống nhau *pop-size* (của điểm khởi đầu cho lần lặp thứ nhất và của điểm tốt nhất được giữ lại cho những lần lặp kế tiếp); nhiều toán tử đột biến khác nhau tạo nên sự đa dạng cho quần thể vào giai đoạn đầu của tiến trình. Khi GENOCOP hội tụ, cá thể x^* tốt nhất được giữ lại và dùng làm điểm khởi đầu x_s cho lần lặp kế tiếp. Nhưng, lần lặp kế tiếp được thực thi với giá trị tham số thưởng phạt giảm ($\tau \leftarrow g(\tau, t)$) và một tập các ràng buộc hiện hành A mới:

$$A \leftarrow A - S \cup V,$$

trong đó, S và V là những tập con của N_i. S là tập các ràng buộc x^* thỏa còn V là tập ràng buộc mà x^* vi phạm. Chú ý rằng việc giảm τ dẫn đến việc tăng thưởng phạt.

Cơ chế của thuật giải được minh họa trong thí dụ sau.

Bài toán là:

Min $f(x) = x_1 * x_2^2$,

c_1: $2 - x_1^2 - x_2^2 \geq 10$.

Lời giải toàn cục lý thuyết là $x^* = (-0.816497, -1.154701)$, và $f(x^*) = -1.088662$. Điểm khởi tạo $x_0 = (-0.99, -0.99)$ thỏa mãn các ràng buộc. Sau lần lặp thứ nhất (A rỗng) hệ thống hội tụ vào $x_1 = (-1.5, -1.5)$, $f(x_1) = -3.375$. Điểm x_1 vi phạm ràng buộc c_1 làm nó trở thành hiện hành. Điểm x_1 được dùng làm điểm khởi đầu cho lần lặp thứ 2. Lần lặp thứ 2 ($\tau = 10^{-1}$, $A = \{c_1\}$) đưa đến $x_2 = (-0.831595, -1.179690)$, $f(x_2) = -1.122678$. Điểm x_2 được dùng làm điểm khởi đầu cho lần lặp thứ 3. Lần lặp thứ 3($\tau = 10^{-2}$, $A = \{c_1\}$) đưa đến $x_3 = (-0.815862, 1.158801)$, $f(x_3) = -1.09985$. Chuỗi các điểm x_t (trong đó $t = 4, 5, ...$ là số lần lặp của thuật giải) hội tụ về điểm tối ưu.

6.3. Các kỹ thuật khác

Trong những năm gần đây, đã có nhiều phương pháp được đề xuất để xử lý các ràng buộc của thuật giải di truyền đối giải bài toán NLP. Hầu hết đều dựa trên khái niệm về hàm thưởng–phạt để phạt những lời giải không khả thi, nghĩa là:

$$eval(x) = \begin{cases} f(x), & \text{nếu } x \text{ thỏa} \\ f(x) + penalty(x), & \text{ngược lại} \end{cases}$$

trong đó, *penalty* (x) = 0, nếu x không vi phạm ràng buộc nào, và *penalty* (x) > 0 ngược lại. Trong hầu hết các phương pháp, một tập các hàm f_j ($1 \leq j \leq p$) được sử dụng để xây dựng thưởng phạt; hàm f_j đo mức vi phạm của ràng buộc thứ j như sau:

$$f_j(x) = \begin{cases} max\{0, g_j(x)\}, & 1 \leq j \leq p \\ |h_j(x)|, & p+1 \leq j \leq m \end{cases}$$

Tuy nhiên, những phương pháp này khác nhau ở nhiều chi tiết quan trọng, cách thiết kế hàm thưởng phạt và cách áp dụng cho những lời giải không thỏa mãn ràng buộc. Ta sẽ lần lượt bàn chi tiết về một số phương pháp như vậy; các phương pháp được trình bày theo thứ tự số tham số cần thiết ít dần.

PHƯƠNG PHÁP #1

Phương pháp này do Homaifar và cộng sự đề nghị. Phương pháp này giả định rằng đối với mỗi ràng buộc ta thiết lập một họ các khoảng để xác định hệ số thưởng phạt thích hợp. Cách thực hiện như sau:

- Đối với mỗi ràng buộc, tạo nhiều cấp độ (l) vi phạm,
- Với mỗi cấp độ vi phạm và mỗi ràng buộc, tạo một hệ số thưởng phạt R_{ij} ($i = 1, 2,..., l$; $j = 1, 2,.., m$); những cấp độ vi phạm cao hơn cần các giá trị lớn hơn cho hệ số này.
- Khởi đầu với một quần thể ngẫu nhiên các cá thể (thỏa mãn lẫn vi phạm ràng buộc),
- Tiến hóa quần thể; lượng giá các cá thể theo công thức:

$$eval(x) = f(x) + \sum_{j=1}^{m} R_{ij} f_j^2(x)$$

Điểm yếu của phương pháp này là trong số các tham số: với m ràng buộc, phương pháp cần m tham số để thiết lập số quãng cho từng ràng buộc (Homaifar sử dụng cùng một tham số cho mọi ràng buộc và bằng l= 4), thêm l tham số cho mỗi ràng buộc (nghĩa là, tổng cộng có l*m tham số; những tham số này biểu diễn các hệ số thưởng phạt R_{ij}). Như vậy phương pháp này cần m($2l$ + 1) tham số để xử lý m ràng buộc. Ví dụ, có m = 5 ràng buộc và l = 4 cấp độ vi phạm, ta cần 5 * (2*4 + 1) = 45 tham số ! Rõ ràng các kết quả và tham số

phụ thuộc nhau. Và thường, đối với một bài toán cho trước, thường chỉ có một tập các tham số tối ưu duy nhất mà hệ thống phải tìm để đạt được lời giải gần – tối ưu nhất, nhưng thật khó mà tìm được tập tham số tối ưu này.

PHƯƠNG PHÁP #2

Phương pháp này được Joines và Houck đề nghị. Trái với phương pháp trên, các tác giả thực hiện thưởng phạt động. Các cá thể được lượng giá (tại lần lặp t) theo công thức:

$$eval(x) = f(x) + (c \times t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(x)$$

trong đó, C, α và β là những hằng số. Các tác giả chọn C= 0.5, $\alpha = \beta = 2$. Phương pháp cần ít tham số hơn nhiều so với phương pháp trước. Cũng vậy, thay vì định nghĩa nhiều cấp độ vi phạm, áp lực trên những lời giải không thỏa mãn tăng lên do $(C*t)*\alpha$ thành phần của số hạng thưởng phạt: về cuối tiến trình (khi thế hệ t lớn), thành phần này chấp nhận các giá trị lớn.

PHƯƠNG PHÁP #3

Phương pháp này do Schoenauer và Xanthakis đề nghị; thực hiện như sau:

- Bắt đầu với một quần thể ngẫu nhiên các cá thể (thỏa mãn hay vi phạm ràng buộc),
- Khởi tạo $j = 1$ (j là biến đếm ràng buộc),
- Tiến hóa quần thể này với $eval(x) = f_j(x)$, cho đến khi một tỉ lệ phần trăm cho trước của quần thể (được gọi là ngưỡng lệch ϕ) thỏa ràng buộc này,

- Tăng $j : j = j + 1$
- Quần thể hiện hành là điểm khởi đầu cho giai đoạn kế tiếp của tiến trình, trong đó, $eval(x) = f_j(x)$. Trong giai đoạn này, những điểm không thỏa một trong các ràng buộc thứ nhất, thứ hai, ..., hay thứ $(j-1)$ sẽ bị loại khỏi quần thể. Tiêu chí dừng lần nữa lại là thỏa ràng buộc thứ j theo phần trăm ngưỡng lệch ϕ của quần thể.
- Nếu $j < m$, lặp lại hai bước sau cùng, ngược lại ($j = m$) tối ưu hóa hàm mục tiêu, nghĩa là, $eval(x) = f(x)$, loại bỏ các cá thể không khả thi.

Phương pháp này đòi hỏi các ràng buộc có bậc tuyến tính được xử lý lần lượt. Ta không thấy rõ tác động của bậc trong các ràng buộc đối với kết quả của các thuật giải; nhưng kinh nghiệm cho thấy các bậc khác nhau cung cấp các kết quả khác nhau (khác về ý nghĩa của tổng thời gian chạy và độ chính xác).

Tổng cộng, phương pháp này cần ba tham số: thừa số chia sẻ σ, ngưỡng lệch ϕ, và bậc cụ thể của các ràng buộc. Phương pháp này khác hai phương pháp trước nhiều, và, nói chung, cũng khác với những phương pháp thưởng phạt khác, vì nó chỉ xét mỗi lần một ràng buộc. Cũng vậy, ở bước cuối cùng của thuật giải, phương pháp này tối ưu hóa hàm mục tiêu f mà không có thành phần thưởng phạt nào.

PHƯƠNG PHÁP #4

GENOCOP II là phương pháp #4. Như đã nói ở trên, đây là phương pháp duy nhất có phân biệt giữa ràng buộc tuyến tính và ràng buộc phi tuyến. Thuật giải duy trì tính khả thi của tất cả các ràng buộc tuyến tính bằng một tập các toán tử đóng, chuyển một lời giải thỏa ràng buộc (thỏa theo nghĩa các ràng buộc tuyến tính mà thôi) thành một lời giải thỏa mãn các ràng buộc khác. Tại mỗi lần



lặp, thuật giải chỉ xét các ràng buộc đang kích hoạt; áp lực trên các lời giải không thỏa mãn tăng lên do các giá trị nhiệt độ τ giảm.

Phương pháp này có thêm một tính chất độc đáo: nó bắt đầu từ một điểm duy nhất, do đó tương đối dễ so sánh nó với những phương pháp tối ưu cổ điển khác. Những phương pháp này được kiểm tra kết quả (đối với một bài toán cho trước) từ một điểm khởi đầu nào đó.

Phương pháp #4 cần nhiệt độ ban đầu τ_0 và nhiệt độ 'đông' τ_f, và một phương án làm lạnh để giảm nhiệt độ τ. Các giá trị chuẩn là $\tau_0 = 1$, $\tau_{i+1} = 0.1*\tau_i$, với $\tau_i = 0.000001$.

PHƯƠNG PHÁP #5

Do Powell đề nghị, phương pháp này là phương pháp thưởng phạt cổ điển với một ngoại lệ đáng chú ý, mỗi cá thể được lượng giá bằng công thức:

$$eval(x) = f(x) + r\sum_{j=1}^{m} f_j(x) + \lambda\ (t,x)$$

trong đó, r là hằng số; nhưng cũng có một thành phần $\lambda\ (t, x)$. Đây là lần lặp bổ sung phụ thuộc hàm có ảnh hưởng đến việc lượng giá những lời giải không thỏa mãn. Vấn đề là phương pháp phân biệt các cá thể thỏa với các cá thể không thỏa ràng buộc theo một heuristic bổ sung: đối với một cá thể thỏa mãn x bất kỳ và một cá thể không thỏa mãn y bất kỳ, $eval(x) < eval(y)$, nghĩa là lời giải thỏa ràng buộc tốt hơn lời giải không thỏa. Có thể đạt được điều này bằng nhiều cách; một khả năng là thiết lập:

$$\lambda\ (t,x) = \begin{cases} 0, & x \in F \\ max(0, max_{x\in F}(f(x)) - min_{x\notin F}(r\sum_{j=1}^{m} f_j(x))), & x \notin F \end{cases}$$



trong đó, F biểu thị phần khả thi của không gian tìm kiếm. Nói cách khác, các cá thể không thỏa mãn bị phạt: các giá trị của chúng không thể tốt hơn giá trị của cá thể thỏa mãn xấu nhất (nghĩa là: $\max_{x\in F}\{f(x)\}$).

PHƯƠNG PHÁP #6

Phương pháp cuối cùng loại bỏ những cá thể không thỏa mãn (phạt chết); phương pháp đã được dùng trong phương pháp chiến lược tiến hóa (xem phụ lục 2), chương trình tiến hóa giải bài toán tối ưu hóa số, và mô phỏng luyện thép.

6.4. Các khả năng khác

Như đã trình bày, nhiều nhà nghiên cứu đã nghiên các heuristic khi thiết kế các hàm thưởng phạt. Một số giả thiết đã được hình thức hoá như sau:

- Thưởng phạt là các hàm tính khoảng cách thỏa mãn ràng buộc sẽ thực hiện tốt hơn các thưởng phạt chỉ đơn thuần là các hàm tính số ràng buộc bị vi phạm.
- Đối với những bài toán ít ràng buộc, và ít lời giải đầy đủ, nếu thưởng phạt chỉ đơn thuần là các hàm tính số ràng buộc bị xâm phạm sẽ khó tìm được lời giải,
- Hàm thưởng phạt tốt có thể được xây dựng từ hai đại lượng, *chi phí hoàn thành cực đại* và *chi phí hoàn thành ước tính*.
- Thưởng phạt cần sát với chi phí hoàn thành ước tính, nhưng thường không nên nhỏ hơn. Thưởng phạt càng chính xác thì lời giải tìm được càng tốt. Khi sự thưởng phạt lượng giá chi phí hoàn thành thường xuyên thấp hơn, thì việc tìm kiếm lời giải thất bại.

Hoặc heuristic khác:

- Thuật giải di truyền có hệ số thưởng phạt thay đổi được sẽ thành công hơn thuật giải có nhân tố thưởng phạt cố định,

Ở đây, sự biến đổi của hệ số thưởng phạt được quyết định theo heuristic.

Quan sát cuối cùng này được Smith và Tate nghiên cứu sâu hơn. Họ đã thử nghiệm với các thưởng phạt động mà phép đo thưởng phạt phụ thuộc số ràng buộc bị xâm phạm, hàm mục tiêu khả thi tốt nhất tìm được cũng như giá trị tốt nhất của hàm này.

Cũng vậy, một phương pháp áp dụng thưởng phạt thích nghi được Bean và Hadj–Alouane đề nghị. Nó dùng một hàm thưởng phạt, một thành phần của hàm thưởng phạt nhận một hồi tác từ tiến trình tìm kiếm. Mỗi cá thể được lượng giá bằng công thức:

$$eval(\overline{X}) = f(\overline{X}) + \lambda(t)\sum_{j=1}^{m} f_j^2(\overline{X}),$$

trong đó, $\lambda(t)$ được cập nhật tại mỗi thế hệ t theo cách:

$$\lambda(t+1) = \begin{cases} (1/\beta_1)\lambda(t), & \overline{B}(i) \in F : \forall t-k+1 \le i \le t \\ \beta_2\lambda(t), & \overline{B}(i) \notin F : \forall t-k+1 \le i \le t \\ \lambda(t), & \text{ngược lại} \end{cases}$$

trong đó, $\overline{B}(i)$ biểu diễn cá thể tốt nhất, trong thế hệ i, β_1, $\beta_2 > 1$ và $\beta_1 \neq \beta_2$ để tránh chu trình. Nói cách khác, phương pháp (1) giảm phạt $\lambda(t+1)$ đối với thế hệ t+1, nếu tất cả các cá thể tốt nhất trong các thế hệ cuối k thỏa mãn, và (2) tăng phạt nếu tất cả các cá thể tốt nhất trong các thế hệ cuối k là không thỏa mãn. Nếu có một số cá thể

thỏa mãn và một số khác không thỏa mãn là những cá thể tốt nhất trong các thế hệ cuối k, λ(t+1) vẫn giữ không đổi.

Phương pháp trên được áp dụng cho những bài toán qui hoạch nguyên. Ví dụ:

Min cx

$Ax - b \ge 0$, (1)

$$\sum_{j=1}^{n_i} x_{ij} = 1, \quad i = 1,2,\dots,m \ (2)$$

$x_{ij} \in \{0,1\}$ (3)

Đối với mỗi tập, i=1,..., m, các phương trình (2) và (3) cần chính xác một biến trong $\{x_{ij}\}_{j=1}^{n_i}$ là 1. Ma trận A là $k*n$ ($n = \sum i = 1^m n_i$) và b là một vectơ hằng k– chiều.

Hầu hết các kỹ thuật giải bài toán trên thành công đều là phương pháp nhánh cận dùng hoặc qui hoạch tuyến tính, hoặc phương pháp Lagrange mở rộng hoặc các biến thể của nó. Phương pháp Lagrange mở rộng loại bớt một số ràng buộc bằng cách tổ hợp một thưởng phạt tuyến tính có trọng vào một vi phạm ràng buộc. Các trọng "đúng" có thể dẫn đến những giới hạn rất tốt hay ngay cả những lời giải tối ưu cho bài toán gốc; một phương pháp Lagrange mở rộng tiêu biểu thay bài toán gốc (các ràng buộc (1)) bằng công thức sau:

Min $cx - \lambda(Ax - b)$

$Ex = e_m$, $x_{ij} \in \{0,1\}$

(các ràng buộc $Ex = e_m$ là những ràng buộc có nhiều lựa chọn, trong đó, e_m là một trong các vectơ đó).

Cách tiếp cận được đề nghị thay bài toán gốc bằng:

Min $cx + p_\lambda(x)$,

$Ex = e_m$, $x_{ij} \in \{0,1\}$

trong đó, $p_\lambda(x) = \sum_{i=1}^{k} \lambda_i [0, A_i x - b_i]^2$. Đây là hàm phi tuyến và một thuật giải di truyền được sử dụng để tối ưu hóa biểu thức.

Có nhiều ý kiến thú vị về phương pháp được đề nghị này. Trước tiên, các kết quả thực nghiệm cho thấy việc khởi đầu với những giá trị λ lớn sẽ không giúp thuật giải thành công. Vì thế thuật giải được đề nghị điều chỉnh vectơ λ đang khi chạy: dùng một chuỗi các vectơ λ tăng dần. Các thử nghiệm cho thấy tốc độ tăng của chuỗi này cực kỳ quan trọng: tốc độ chậm có thể cải thiện chất lượng lời giải, còn tốc độ nhanh có thể làm tiến hóa di truyền không hiệu quả. Hơn nữa, thuật giải di truyền được đề nghị dùng một kỹ thuật khóa ngẫu nhiên, mà lời giải được biểu diễn bằng vectơ các số ngẫu nhiên: thứ tự sắp xếp của chúng giải thích lời giải (chú ý sự tương tự giữa phương pháp này và ứng dụng chiến lược tiến hóa trong bài toán người du lịch; xem phụ lục 2 về chiến lược tiến hóa và chương 8 về bài toán người du lịch). Lời giải được biểu diễn bằng một chuỗi có chiều dài bằng số các tập nhiều chọn lựa. Mỗi vị trí i trong chuỗi có thể có giá trị nguyên trong $\{1,..., n_i\}$, với n_i là số các biến trong tập nhiều chọn lựa.

Các phương pháp xử lý ràng buộc khác cũng đáng được lưu tâm. Một trong số đó dựa trên thuật giải sửa chữa: một lời giải không thỏa mãn x bị "buộc" vào vùng thỏa mãn và được lượng giá theo phiên bản đã sửa chữa của nó. Cũng có thể xây dựng thuật giải lai kết hợp một số thủ tục tối ưu hóa tất định.

Một khả năng nữa là việc dùng các giá trị của hàm mục tiêu f và các thưởng phạt f_j làm thành phần của một vectơ và áp dụng các kỹ thuật đa–mục tiêu để cực tiểu hóa tất cả các thành phần của vectơ này. Nói cách khác, hàm mục tiêu f và các độ đo vi phạm ràng buộc f_j (với m ràng buộc) tạo nên vectơ v có $a*(m+1)$ chiều:

$v = (f, f_1, ..., f_m)$.

Bằng cách dùng một phương pháp tối ưu hóa đa mục tiêu nào đó; ta có thể cực tiểu hóa các thành phần của nó: một lời giải lý tưởng x có thể có $f_j(x) = 0$ với $1 \le j \le m$ và $f(x) \le f(y)$ với mọi y thỏa mãn (các bài toán cực tiểu hóa).

Nhưng có một phương pháp khác được đề nghị bởi Le Riche và cộng sự cũng đáng quan tâm. Các tác giả thiết kế một thuật giải di truyền tách biệt dùng hai giá trị tham số thưởng phạt (cho mỗi ràng buộc) thay vì chỉ một; hai giá trị này có mục đích đạt được cân bằng giữa các thưởng phạt nặng và trung bình bằng các duy trì hai quần thể con của các cá thể. Quần thể bị phân thành hai nhóm hợp tác, mà các cá thể trong mỗi nhóm được lượng giá bằng một trong hai tham số thưởng phạt.

Cũng cần nhắc đến một phương pháp thú vị khác được Paredis báo cáo gần đây. Phương pháp này (được mô tả trong ngữ cảnh các

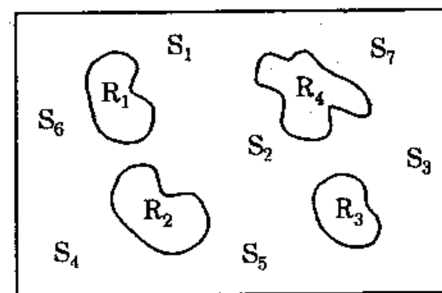
bài toán thỏa ràng buộc) dựa trên mô hình đồng – tiến hóa, theo đó quần thể các lời giải đồng tiến hóa với quần thể các ràng buộc: các lời giải thích hợp hơn thỏa nhiều ràng buộc hơn, trong khi những ràng buộc thích hợp hơn bị nhiều lời giải vi phạm hơn. Điều này có nghĩa là các cá thể trong quần thể lời giải được xét từ toàn bộ không gian tìm kiếm, và có nghĩa là không có phân biệt giữa những cá thể thỏa mãn và không thỏa mãn. Nhưng việc lượng giá một cá thể được quyết định trên cơ sở các phép đo vi phạm ràng buộc f_j ; những f_j tốt hơn (các ràng buộc hoạt động) có thể đóng góp nhiều hơn vào việc lượng giá lời giải. Có thể khá hay nếu áp dụng phương pháp này vào những bài toán tối ưu số có ràng buộc và so sánh nó với những phương pháp khác, nhưng khó khăn chính phải giải quyết trong việc áp dụng như thế dường như cũng rất giống với những phương pháp khác: làm sao để cân đối áp lực về tính thỏa mãn của lời giải với áp lực cực tiểu hóa hàm mục tiêu.

6.5. GENOCOP III

Phương pháp này kết hợp hệ GENOCOP (mô tả trong phần 6.1, nhưng cũng mở rộng nó bằng cách duy trì hai quần thể riêng biệt, ở đây việc phát triển trong một quần thể ảnh hưởng đến các tiến hóa của các cá thể trong quần thể khác. Quần thể thứ nhất *Ps* gồm các điểm tìm kiếm thỏa các ràng buộc tuyến tính của bài toán (giống như trong GENOCOP). Tính thỏa mãn (theo nghĩa các ràng buộc tuyến tính) của các điểm này được duy trì, giống như trước, bằng các toán tử chuyên biệt hóa. Quần thể thứ hai *Pr* gồm các điểm tham chiếu; các điểm này hoàn toàn thỏa mãn, nghĩa là, chúng thỏa mọi ràng buộc (nếu GENOCOP III gặp khó khăn trong việc định vị những điểm tham chiếu nhằm mục đích tối ưu hóa như thế, nó sẽ

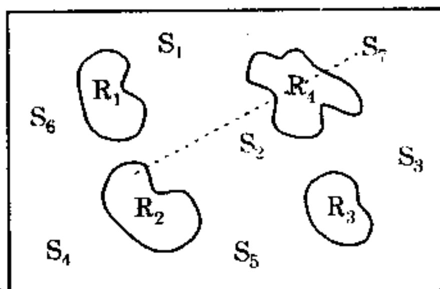
nhắc người sử dụng. Trong các trường hợp tỉ lệ ρ, giữa kích thước thỏa mãn và toàn bộ không gian tìm kiếm, là rất nhỏ, có thể sẽ xảy ra việc tập hợp khởi tạo của các điểm tham chiếu cần có nhiều bản sao của chỉ một điểm thỏa mãn). Hình 6.3 minh họa hai quần thể.

Các điểm tham chiếu $\overline{R}$, thuộc loại thỏa mãn được lượng giá trực tiếp bằng hàm mục tiêu (nghĩa là, $eval(\overline{R})=f(\overline{R})$). Mặt khác, các điểm tìm kiếm không thỏa mãn được "sửa chữa" cho việc tiến hóa và tiến trình sửa chữa làm việc như sau. Giả sử có điểm tìm kiếm $\overline{S}$ không hoàn toàn thỏa mãn. Trong trường hợp đó, hệ thống chọn (những điểm tham chiếu tốt hơn có cơ hội được chọn cao hơn; một phương pháp chọn dựa trên xếp hạng phi tuyến được dùng) một trong những điểm tham chiếu, như $\overline{R}$, và tạo những điểm ngẫu nhiên $\overline{Z}$ từ một đoạn giữa $\overline{S}$ và $\overline{R}$ bằng cách phát sinh các số ngẫu nhiên a trong khoảng <0,1> : $\overline{Z} = a\overline{S}+(1-a)\overline{R}$. Hình 6.4 minh họa tiến trình sửa chữa này.



Hình 6.3. Quần thể Ps = $\{\overline{S}_1, \overline{S}_2, \overline{S}_3, \overline{S}_4\}$

và quần thể Pr = $\{\overline{R}_1, \overline{R}_2, \overline{R}_3, \overline{R}_4\}$



Hình 6.4. Lượng giá của điểm $\overline{S}$ không khả thi.

Một khi đã tìm được $\overline{Z}$ thỏa mãn, $eval(\overline{S}) = eval(\overline{Z}) = f(\overline{Z})$. Hơn nữa, nếu $f(\overline{Z})$ tốt hơn $f(\overline{R})$, thì điểm $\overline{Z}$ thay $\overline{R}$ làm điểm tham chiếu mới. $\overline{Z}$ cũng thay $\overline{S}$ với xác suất thay thế *Pr* cho trước.

GENOCOP III tránh được nhiều bất lợi của những hệ thống khác. Nó chỉ đưa ra một ít tham số bổ sung (kích thước quần thể của các điểm tham chiếu, xác suất thay thế). Nó luôn luôn trả về lời giải thỏa mãn. Một không gian tìm kiếm thỏa mãn được tìm bằng cách tạo những tham chiếu từ những điểm tìm kiếm. Lân cận của những điểm tham chiếu tốt thường được khảo sát nhiều hơn. Một số điểm tham chiếu được chuyển vào quần thể các điểm tìm kiếm, ở đó chúng được biến đổi bởi các toán tử chuyên biệt (bảo tồn các ràng buộc tuyến tính). Thực nghiệm cũng chứng tỏ GENOCOP III thường hiệu quả hơn các phương pháp khác.

Chương 7

BÀI TOÁN VẬN TẢI

Trong chương 6, chúng tôi đã trình bày các cách tiếp cận khác nhau hỗ trợ GA khi xử lý các ràng buộc. Tuy nhiên, với những bài toán đặc biệt (như bài toán vận tải chẳng hạn) nếu biết tận dụng những tri thức về bài toán, ta có thể thực hiện tốt hơn: sử dụng một cấu trúc dữ liệu (tự nhiên) thích hợp hơn (cho bài toán vận tải, một ma trận) và các toán tử di truyền chuyên biệt thao tác trên ma trận. Một tiến trình tiến hóa như thế, sẽ là một phương pháp mạnh hơn phương pháp mô tả trong chương trước nhiều. Các chương trình tiến hóa cài đặt các phương pháp xử lý ràng buộc của chương 6 giải bài toán có thể tối ưu có ràng buộc tổng quát trong khi chương trình tiến hóa trình bày trong chương 7 này chỉ thích hợp cho bài toán vận tải. Khi trình bày bài toán vận tải, chúng tôi chỉ muốn nhấn mạnh rằng tri thức về bài toán có thể giúp ích nhiều cho quá trình giải nó, nghĩa là, với một cấu trúc dữ liệu thích hợp cộng với các phép toán di truyền chuyên biệt ta sẽ có một chương trình tiến hóa tốt. Đây cũng chính là mục đích của cuốn sách.

7.1. Bài toán vận tải tuyến tính

Bài toán vận tải là một trong những bài toán tối ưu có ràng buộc đã được nghiên cứu chi tiết trong nhiều công trình. Mục đích là xây dựng một dự án vận tải với chi phí thấp nhất để vận chuyển một loại hàng hóa, từ một số nguồn đến một số đích. Bài toán yêu cầu cho biết khả năng cung ứng tại từng nguồn, nhu cầu của mỗi đích, và chi phí vận chuyển từ mỗi nguồn đến mỗi đích.

Do chỉ có một loại hàng hóa, mỗi đích có thể nhận hàng từ một hay nhiều nguồn. Mục tiêu là tìm số lượng cần chuyên chở từ mỗi nguồn đến mỗi đích sao cho tổng chi phí chuyên chở là tối thiểu.

Bài toán vận tải là tuyến tính nếu chi phí tỉ lệ với số lượng hàng vận tải; nếu không, nó là phi tuyến.

Giả sử có n nguồn và k đích. Số cung tại nguồn i là *sour(i)* và số cầu tại đích j là *dest(j)*. Chi phí vận tải một đơn vị hàng giữa nguồn i và đích j là *cost(i,j)*.

Nếu x_{ij} là số lượng hàng được vận tải từ nguồn i đến đích j thì bài toán vận tải đã cho là bài toán tối ưu:

$$\text{Min } \sum_{i=1}^{n}\sum_{j=1}^{k} f_{ij}(x_{ij})$$

$$\sum_{j=1}^{k} x_{ij} \le sour(i), \quad i = 1,2,...,n$$
$$\sum_{i=1}^{n} x_{ij} \ge dest(j), \quad j = 1,2,...,k$$
$$x_{ij}, \quad i = 1,2,...,n; \quad j = 1,2,...,k$$

Tập các ràng buộc thứ nhất quy định rằng số lượng hàng được chở từ một nguồn không thể vượt quá số cung của nó; tập thứ hai yêu cầu rằng số lượng hàng chở đến đích phải thỏa số cầu của nó. Nếu $f_{ij}(x_{ij}) = cost(ij)*x_{ij}$ với mọi i, j, bài toán là tuyến tính.

Bài toán trên bao hàm rằng tổng số cung $\sum_{i=1}^{k} sour(i)$ ít nhất phải bằng với tổng số cầu $\sum_{j=1}^{n} dest(j)$. Khi tổng số cung bằng tổng số cầu, bài toán vận tải gọi là *bài toán vận tải cân bằng*. *Bài toán vận tải cân bằng* chỉ khác bài toán vận tải tuyến tính ở chỗ là tất cả các ràng buộc tương ứng là các phương trình; nghĩa là

$$\sum_{j=1}^{k} x_{ij} = sour(i), \quad i = 1,2,...,n$$

$$\sum_{i=1}^{n} x_{ij} = dest(j), \quad j = 1,2,...,k$$

Nếu tất cả *sour(i)* và *dest(j)* là các số nguyên, một lời giải tối ưu bất kỳ của bài toán vận tải tuyến tính cân bằng cũng là lời giải nguyên, nghĩa là tất cả x_{ij} (i = 1, 2, 3,..., n), (j = 1, 2, 3,..., k) là những số nguyên. Hơn nữa, số lượng các số nguyên dương x_{ij}, tố đa là $k + n - 1$; trong phần này chúng tôi trình bày bài toán vận tải tuyến tính cân bằng.

Thí dụ 7.1. Giả sử có 3 nguồn và 4 đích, số cung tương ứng là:

sour(1) =15, *sour*(2)= 25 và *sour*(3)=5.

Số cầu tương ứng là:

dest(1)=5, *dest*(2) =15 *dest* (3)= 15 và *dest*(4)=10.

Chú ý là tổng số cung và tổng số cầu đều bằng 45.

Chi phí vận chuyển một đơn vị, *cost(i, j)* (i = 1, 2, 3 và j=1, 2, 3, 4) được cho trong bảng sau:

Chi phí

10	0	20	11
12	7	9	20
0	14	16	18

Lời giải tối ưu được trình bày trong bảng dưới. Tổng chi phí là 315. Lời giải gồm các giá trị nguyên của x_{ij}.

Số lượng vận tải

	5	15	15	10
15	0	5	0	10
25	0	10	15	0
5	5	0	0	0

7.1.1. Thuật giải di truyền cổ điển giải bài toán vận tải cân bằng

Khi nói thuật giải di truyền "cổ điển", chúng tôi muốn nói đến một thuật giải mà trong đó các nhiễm sắc thể (nghĩa là, cách biểu diễn lời giải) là những chuỗi bit - danh sách các số 0 và 1. Một phương pháp dễ hiểu để định nghĩa một vectơ bit cho 1 lời giải bài toán vận tải là tạo một vectơ $<v_1,v_2,...,v_p>$ ($p = n * k$), sao cho mỗi thành phần v_i ($i = 1, 2,..., p$), là một vectơ bit $<w_0^i, ..., w_s^i>$ và biểu diễn một số nguyên liên quan đến hàng j và cột m trong ma trận phân phối, mà ở đây $j = \lfloor(i\text{-}1)/k + 1\rfloor$ và $m = (i\text{-}1) \bmod k + 1$. Chiều dài của các vectơ w (tham số f) quyết định số nguyên lớn nhất (2^{s+1}-1) cần có để biểu diễn một lời giải.

Ta sẽ thảo luận một cách ngắn gọn về cách biểu diễn này và việc phải thỏa mãn ràng buộc, cũng như về hàm lượng giá và các toán tử di truyền.

Thỏa mãn ràng buộc: Rõ ràng là mỗi vectơ lời giải phải thỏa các ràng buộc sau đây:

$$\sum_{i=ck+1}^{ck+k} v_i = sour[c+1], \quad c = 0,1,2,...,n-1$$
$$\sum_{j=m,\,step\,k}^{kn} v_j = dest[m], \quad m = 1,2,...,k$$
$$v_q \geq 0, \quad q = 1,2,...,n; \quad j = 1,2,...,k-n$$

Chú ý là ràng buộc cuối cùng luôn được thỏa (ta thông dịch một chuỗi các số 0 và 1 là một số nguyên dương). Hai ràng buộc đầu thể hiện tổng cung và tổng cầu tại mỗi nguồn và mỗi đích, mặc dù những công thức này không đối xứng.

Hàm lượng giá: Hàm lượng giá đơn giản là tính tổng chi phí vận tải từ các nguồn đến các đích và được tính theo:

$$eval(<v_1,v_2,...,v_p>) = \sum_{i=1}^{p} v_i . cost[j][m]$$

trong đó, $j = \lfloor(i\text{-}1)/k + 1\rfloor$ và $m = (i\text{-}1) \bmod k + 1$.

Các toán tử di truyền: Không có định nghĩa tự nhiên nào về các thông tin di truyền đối với bài toán vận tải trong biểu diễn như trên. Đột biến thường được định nghĩa là thay đổi của một bit trong một vectơ lời giải. Điều này tương ứng với thay đổi của một giá trị nguyên v_i. Đối với những bài toán của ta, đột biến như vậy lại có thể gây ra một chuỗi những thay đổi ở những điểm khác nhau (ít nhất là ba thay đổi liên quan) để duy trì các ràng buộc. Chú ý là ta luôn phải nhớ thay đổi đã xảy ra trong cột nào và hàng nào - mặc dù là biểu diễn vectơ nhưng ta suy nghĩ và thực hiện theo cách các hàng và các cột (các nguồn và các đích). Đây là lý do khiến công thức trở nên phức tạp; dấu hiệu đầu tiên của tính phức tạp này là mất đi tính đối xứng khi diễn tả các ràng buộc.

Cũng có một số thắc mắc khác đặt ra. Đột biến được hiểu là những thay đổi nhỏ trong vectơ lời giải, nhưng theo những gì ta biết trước đây, chỉ một thay đổi trong một số nguyên lại có thể gây ra ít nhất ba thay đổi khác tại những vị trí thích hợp. Giả sử hai điểm ngẫu nhiên (v_i và v_m, với $i < m$) được chọn sao cho chúng thuộc cùng một hàng hay một cột. Ta hãy giả sử rằng v_i, v_j, v_k, v_m, ($i < j < k < m$) là những thành phần của vectơ lời giải (được chọn để đột biến)

sao cho v_i và v_k cũng như v_j và v_m thuộc một cột, và v_i, v_j cũng như v_k, v_m thuộc cùng một hàng.

Tức là, trong biểu diễn ma trận, ta có:

... .

... v_i ... v_j ...

... .

... .

... v_k ... v_m ...

... .

Bây giờ, ta gặp một khó khăn khi phải xác định thay đổi cần thiết trong vectơ lời giải. Ta nên tăng hay giảm v_i ? Ta có thể chọn cách đổi nó thành một (thay đổi nhỏ nhất có thể) hay thành một số ngẫu nhiên nào đó trong khoảng <0, 1,.., v_i>. Nếu tăng giá trị v_i theo một hằng số C ta phải giảm mỗi giá trị v_j và v_k theo cùng hằng số đó. Điều gì xảy ra nếu $v_j < C$ hoặc $v_k < C$? Ta có thể chọn $C = min(v_i, v_j, v_k)$, nhưng như thế, phần lớn các đột biến sẽ có kết quả không thay đổi gì cả, do xác suất của việc chọn ba phần tử không là 0 có thể gần bằng 0 (nhỏ hơn $1/n$ đối với các vectơ có kích thước n^2).

Như vậy phương pháp thay đổi 1 bit làm cho các toán tử đột biến không hiệu quả vì cần những biểu thức phức tạp để kiểm soát hàng hay cột tương ứng của phần tử được chọn.

Tình trạng này còn có thể phức tạp hơn nếu ta cố sửa một nhiễm sắc thể sau khi áp dụng toán tử lai. Phá vỡ một vectơ tại một điểm ngẫu nhiên có thể đưa đến việc một cặp nhiễm sắc thể vi phạm nhiều ràng buộc. Nếu ta cố hiệu chỉnh những lời giải này để thỏa tất

cả các ràng buộc, chúng sẽ mất gần hết những tương đồng với cha mẹ. Hơn nữa, cách làm như vậy không rõ ràng chút nào: nếu một vectơ v ở bên ngoài không gian tìm kiếm, "việc sửa đổi" nó cũng khó như giải bài toán gốc. Ngay cả khi thành công trong việc xây dựng một hệ thống dựa trên các thuật giải sửa chữa, thì hệ thống đó cũng khó mà tổng quát hóa được.

Như vậy, biểu diễn vectơ nhị phân như trên không phải là cách thích hợp nhất để định nghĩa những toán tử di truyền trong những bài toán có ràng buộc kiểu này.

7.1.2. Kết hợp tri thức của bài toán để biểu diễn lời giải

Liệu có cách nào biểu diễn lời giải mà vẫn bảo tồn cấu trúc cơ bản của biểu diễn vectơ này khi thực hiện các phép di truyền không ? Ta tin là được, nhưng cần kết hợp tri thức của bài toán vào biểu diễn này.

Trước tiên, chúng tôi mô tả cách tạo một lời giải thỏa tất cả ràng buộc. Ta gọi thủ tục này là **khởi_tạo** - nó sẽ là thành phần cơ bản của toán tử đột biến khi ta bàn đến các toán tử di truyền của các cấu trúc - hai chiều. Nó tạo ra một ma trận có tối đa $k + n - 1$ phần tử khác 0, sao cho tất cả các ràng buộc được thỏa.

Thủ tục khởi tạo

Nhập: mảng *dest[k]*, *sour[n]*;

Xuất: mảng $(v)_{ij}$ sao cho $v_{ij} \geq 0$ với mọi i và j

$$\sum_{j=1}^{k} v_{ij} = dest[i], \quad i = 1, 2, ..., n$$

$\sum_{i=1}^{n} v_{ij} = sour[j], \quad j = 1, 2, \ldots, k$

nghĩa là tất cả các ràng buộc được thỏa.

khởi_tạo;

Bắt đầu

L ← {1,2,...,k*n}: là danh sách các điểm chưa được xét;

Lặp

- chọn một số ngẫu nhiên *q* trong L;
- và *q* được gọi là điểm đã xét
- (hàng) $i \leftarrow \lfloor (q-1)/k + 1 \rfloor$
- (cột) $j \leftarrow (q-1) \bmod k + 1$
- $val \leftarrow min(sour[i], dest[j])$
- $v_{ij} \leftarrow val$
- $sour[i] \leftarrow sour[i] - val$
- $dest[j] \leftarrow dest[j] - val$

Hết lặp nếu (tất cả các điểm trong L đều được thăm)

Kết thúc

Thí dụ 7.2. Với ma trận trong thí dụ 7.1,

nghĩa là:

sour[1] = 15, *sour*[2] =25, *sour*[3] = 5

dest[1] = 5, *dest*[2] =15 , *dest*[3] =15, *dest*[4] = 10

có tất cả 3*4 =12 số, đều chưa được thăm từ đầu. Chọn số ngẫu nhiên đầu tiên, 10 chẳng hạn. Số này được thông dịch thành (hàng) $i = 3$ và (cột) $j = 2$. val = min (*sour*[3], *dest*[2]) = 5, vì thế $v_{32} = 5$. Cũng chú ý rằng sau lần lặp thứ nhất, *sour*[3] = 0 và *dest*[2] = 10.

Ta lặp lại cách tính này với ba số (chưa thăm) ngẫu nhiên, như 8, 5, và 3 (tương ứng với hàng 2 và cột 4, hàng 2 và cột 1, và hàng 1 và cột 3). Ma trận có được v_{ij} (cuối cùng) có nội dung như sau:

	0	**10**	**0**	**0**
0			15	
10	5			10
0		5		

Chú ý rằng các giá trị *sour*[i] và *dest*[j] là những số có được sau 4 lần lặp.

Nếu có thêm chuỗi các số ngẫu nhiên là 1, 11, 4, 12, 7, 6, 9, 2, thì sẽ cho ma trận cuối cùng (với chuỗi các số ngẫu nhiên được chấp nhận <10, 8, 5, 3, 1, 11, 4, 12, 7, 6, 9, 2 >) là:

	0	**0**	**0**	**0**
0	0	0	15	0
0	5	10	0	10
0	0	5	0	0

Rõ ràng, sau 12 lần lặp tất cả (các bản sao cục bộ của) *sour*[i] và *dest*[j] = 0, cũng chú ý rằng, có nhiều chuỗi số mà thủ tục **khởi_tạo** có thể tạo ra lời giải tối ưu cho chúng. Thí dụ, lời giải tối ưu (được cho trong thí dụ 7.1) có thể nhận được đối với bất cứ chuỗi nào sau

đây:<7, 9, 4, 2, 6, *, *, *, *, *, *, * > (trong đó * biểu thị một số chưa thăm bất kỳ), cũng như đối với nhiều chuỗi khác.

Kỹ thuật này có thể phát sinh một lời giải khả thi chứa tối đa $k+n-1$ phần tử nguyên khác 0. Nó sẽ không phát sinh những lời giải khác, có thể cũng khả thi, nhưng không dùng chung đặc trưng này. Thủ tục khởi tạo chắc chắn phải được hiệu chỉnh khi ta định giải *bài toán vận tải phi tuyến*.

Tri thức và những đặc trưng lời giải của bài toán cho ta một cơ hội khác để biểu diễn lời giải của bài toán vận tải bằng một vectơ. Vectơ lời giải sẽ là một chuỗi các số nguyên phân biệt $k * n$ trong khoảng <1, $k * n$ >, (theo thủ tục **khởi_tạo**) sẽ sinh ra một lời giải khả thi. Nói cách khác, ta sẽ xem vectơ lời giải là một hoán vị của các số, và ta có thể tìm các hoán vị cụ thể tương ứng với lời giải tối ưu.

Ta sẽ bàn ngắn gọn về những cài đặt của biểu diễn này trong việc phải thỏa ràng buộc, và về hàm lượng giá cũng như các toán tử di truyền.

Thỏa mãn ràng buộc: Bất cứ hoán vị nào của k*n số phân biệt sẽ sinh ra một lời giải duy nhất thỏa tất cả các ràng buộc, điều này được bảo đảm bởi thủ tục **khởi_tạo.**

Hàm lượng giá: Điều này tương đối dễ: hoán vị nào cũng tương ứng với một ma trận duy nhất, như (v_{ij}) chẳng hạn. Hàm lượng giá là:

$$\sum_{i=1}^{k}\sum_{j=1}^{n} v_{ij}.\cos t[i][j]$$

Các toán tử di truyền: Cũng rất dễ thực hiện.

- Đảo: nếu $<x_1,x_2,...,x_q>$ ($q = k*n$) là một lời giải thì vectơ đảo của nó $<x_q, x_{q-1}, ..., x_1>$ cũng là một lời giải.

- Đột biến: bất cứ hai phần tử nào của vectơ lời giải $<x_1,x_2,...,x_q>$, x_i và x_j chẳng hạn, cũng có thể được hoán vị và vectơ kết quả cũng là một lời giải.
- Lai tạo: hơi phức tạp hơn một chút. Chú ý rằng một toán tử lai (mù quáng) ngẫu nhiên có thể sinh ra các lời giải không hợp lệ: áp dụng một toán tử lai như thế vào các chuỗi: <1,2,3,4,5,6,| 7,8,9,10,11,12> và < 7,3,1,11,4,12,| 5,2,10,9,6,8> sẽ sinh ra (ở đây điểm lai tạo ở sau vị trí thứ 6) các lời giải <1,2,3,4,5,6,5,2,10,9,6,8> và < 7,3,1,11,4,12, 7,8,9,10,11,12>. Cả hai đều là những lời giải không hợp lệ.

Như vậy, ta phải dùng một dạng toán tử lai heuristic nào đó. Có một số tương đồng giữa các chuỗi vectơ lời giải này và các chuỗi vectơ lời giải của bài toán người du lịch. Ở đây ta dùng một toán tử lai heuristic, mà khi cho một cặp cha mẹ, sẽ tạo một con hợp lệ theo thủ tục sau đây:

(1) Tạo bản sao của phần tử thứ hai trong cặp cha-mẹ đã cho ;

(2) Chọn một phần ngẫu nhiên từ phần tử thứ nhất trong cặp cha-mẹ đó;

(3) Tạo những thay đổi tối thiểu cần thiết ở con để có được lời giải hợp lệ.

Thí dụ, nếu cặp cha-mẹ như thí dụ trên, và phần chọn được là (4,5,6,7), thì con có được là < 3,1,11,4,5,6,7,12,2,10,9,8>.

Đúng như ta mong muốn, con có mối liên hệ cấu trúc với cả cha và mẹ. Rồi vai trò của cặp cha-mẹ này có thể được thay đổi khi sinh con thứ hai.

Từ nay, ta sẽ gọi chương trình tiến hóa giải bài toán vận tải tuyến tính cân bằng với biểu diễn vectơ nguyên là hệ GENETIC-1.

7.1.3. Biểu diễn lời giải bài toán vận tải bằng ma trận

Có lẽ biểu diễn lời giải tự nhiên nhất cho bài toán vận tải là cấu trúc hai chiều. Nói cách khác, ma trận V = (v_{ij}) ($1 \le i \le k$, $1 \le j \le n$) có thể biểu diễn lời giải.

Ta sẽ bàn về những cài đặt của biểu diễn ma trận đồng thời cũng diễn tả ràng buộc, và về hàm lượng giá cũng như các toán tử di truyền.

Thỏa mãn ràng buộc: Rõ ràng là mỗi ma trận lời giải $V = (v_{ij})$ có thể thỏa các ràng buộc sau:

$$\sum_{j=1}^{n} v_{ij} = sour[i], \quad i = 1,2,...,k$$

- $$\sum_{i=1}^{k} v_{ij} = dest[j], \quad j = 1,2,...,n$$
  $v_{ij} \ge 0, \quad i = 1,2,...,k; \quad j = 1,2,...,n$

- Điều này tương tự với tập các ràng buộc trong phương pháp dễ hiểu (phần 7.1.2), nhưng các ràng buộc được diễn tả theo cách dễ dàng và tự nhiên hơn.

Hàm lượng giá: hàm lượng giá biểu thị là hàm mục tiêu thông thường:

$$eval(v_{ij}) = \sum_{i=1}^{k}\sum_{j=1}^{n} v_{ij}.\cos t[j][m]$$

Công thức này đơn giản hơn nhiều so với phương pháp dễ hiểu và nhanh hơn trong hệ GENETIC-I, mà mỗi chuỗi phải được biến đổi (khởi tạo) thành một ma trận lời giải trước khi lượng giá.

Các toán tử di truyền: ở đây, ta định nghĩa hai toán tử di truyền, đột biến và lai. Thật khó mà định nghĩa một toán tử đảo có ý nghĩa trong trường hợp này.

Đột biến: giả sử rằng $\{i_1, i_2,..., i_p\}$ là tập con của $\{1, 2, ..., k\}$ và $\{j_1, j_2,..., j_q\}$ là tập con của $\{1, 2, ..., n\}$ sao cho $2 \le p \le k$, $2 \le q \le n$.

Ta hãy biểu thị một cha (mẹ) của đột biến bằng ma trận $(k \times n)$, $V = (v_{ij})$. Rồi tạo một ma trận con $(p \times q)$, $W = (w_{ij})$, từ tất cả các phần tử của ma trận V theo cách sau: một phần tử $v_{ij} \in V$ ở trong W nếu và chỉ nếu $i \in \{i_1, i_2,..., i_p\}$ và $j \in \{j_1, j_2,..., j_q\}$ (nếu $i = i_r$ và $j = j_s$, thì phần tử v_{ij} được đặt trong hàng r và cột s của ma trận W).

Bây giờ ta có thể gán các giá trị mới *sour* $W[i]$và *dest* $W[j]$ ($1 \le i \le p$, $1 \le j \le q$) cho ma trận W:

$$sour \quad W[i] = \sum_{j \in \{j_1, j_2,..., j_q\}} v_{ij}, \quad 1 \le i \le p,$$

$$dest \quad W[j] = \sum_{i \in \{i_1, i_2,..., j_p\}} v_{ij}, \quad 1 \le j \le q.$$

Ta có thể dùng thủ tục **khởi_tạo** (phần 7.1.3) để gán các giá trị mới vào ma trận W sao cho tất cả các ràng buộc *sour* $W[i]$và *dest* $W[j]$ được thỏa. Sau đó, ta thay những phần tử thích hợp của ma trận V bằng các phần tử mới của ma trận W. Bằng cách này , tất cả các ràng buộc toàn cục (*sour*[i]và *dest*[j]) được bảo toàn.

Thí dụ sau đây minh họa toán tử đột biến.

Thí dụ 7.3. Bài toán vận tải có 4 nguồn và 5 đích với những ràng buộc:

sour[1] = 8, *sour*[2] = 4, *sour*[3] = 12, *sour*[4] = 6,

dest[1] = 3, *dest*[2] = 5, *dest*[3] = 10, *dest*[4] = 7, *dest*[5] = 5.

Giả sử rằng ma trận V sau đây được chọn làm cha(mẹ) cho đột biến:

0	0	5	0	3
0	4	0	0	0
0	0	5	7	0
3	1	0	0	2

Chọn (ngẫu nhiên) hai hàng – giả sử là {2,4} – và 3 cột – {2,3,5}. Ma trận con W tương ứng:

4	0	0
1	0	2

Chú ý, *sour* $W[1] = 4$, *sour* $W[2] = 3$, *dest* $W[1] = 5$, *dest* $W[2] = 0$, *dest* $W[3] = 2$. Sau khởi tạo lại của ma trận W, ma trận có những giá trị sau:

2	0	2
1	0	2

Vì thế, cuối cùng con của ma trận V sau đột biến là:

0	0	5	0	3
0	2	0	0	2
0	0	5	7	0
3	3	0	0	0

LAI TẠO

Giả sử rằng hai ma trận $V_1 = (v_{ij}^1)$ và $V_2 = (v_{ij}^2)$ được chọn làm cha-mẹ để thực hiện lai. Ta mô tả thuật giải được dùng để sinh hai con V_3 và V_4.

Tạo hai ma trận tạm: $DIV = (div_{ij})$ và $REM = (rem_{ij})$. Với:

$$div_{ij} = \left\lfloor (v_{ij}^1 + v_{ij}^2)/2 \right\rfloor$$

$$rem_{ij} = (v_{ij}^1 + v_{ij}^2) \bmod 2$$

Ma trận DIV lưu các giá trị trung bình được làm tròn từ cả cha lẫn mẹ, ma trận REM theo dõi xem việc làm tròn nào là cần thiết.

Ma trận REM có một số thuộc tính thú vị sau: số các số 1 trong mỗi hàng và mỗi cột bằng nhau. Nói cách khác, các giá trị của *sour* $REM[i]$ và *dest* $REM[j]$ là những số nguyên chẵn. Ta dùng thuộc tính này để biến đổi REM thành hai ma trận REM_1 và REM_2 sao cho:

$REM = REM_1 + REM_2$,

sour $REM1[i]$ = *sour* $REM2[i]$ = *sour* $REM[i]/2$, $i = 1,\dots,k$.

dest $REM1[j]$ = *dest* $REM2[j]$ = *dest* $REM[j]/2$, $j = 1,\dots, n$.

Rồi sinh ra hai con của V_1 và V_2:

$V_3 = DIV + REM_1$

$V_4 = DIV + REM_2$.

Thí dụ sau đây sẽ minh họa phép lai trên.

Thí dụ 7.4. Lấy lại bài toán trong thí dụ 7.1

Giả sử các ma trận V_1 và V_2 sau đây đã được chọn làm cha-mẹ để lai tạo:

V_1

1	0	0	7	0
0	4	0	0	0
2	1	4	0	5
0	0	6	0	0

V_2

0	0	5	0	3
0	4	0	0	0
0	0	5	7	0
3	1	0	0	2

Các ma trận *DIV* và *REM* là:

DIV

0	0	2	3	1
0	4	0	0	0
1	0	4	3	2
1	0	3	0	1

REM

1	0	1	1	1
0	0	0	0	0
0	1	1	1	1
1	1	0	0	0

Hai ma trận REM_1 và REM_2 là:

REM_1

0	0	1	0	1
0	0	0	0	0
0	1	0	1	0
1	0	0	0	0

REM_2

1	0	0	1	0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0

Cuối cùng, hai con V_3 và V_4 là:

V_3

0	0	3	3	2
0	4	0	0	0
1	1	4	4	2
2	0	3	0	1

V_4

1	0	2	4	1
0	4	0	0	0
1	0	5	3	3
1	1	3	0	1

Chương trình tiến hóa này ta gọi là hệ GENETIC-II.

7.2. Bài toán vận tải phi tuyến

Phần này, ta bàn về chương trình tiến hóa giải bài toán vận tải phi tuyến cân bằng, theo 5 thành phần của thuật giải di truyền: biểu diễn , khởi tạo, lượng giá, các toán tử và các tham số. Hệ thống cũng đặt tên là GENETIC-2 như trong trường hợp tuyến tính.

7.2.1. Biểu diễn

Cũng như trường hợp tuyến tính, ta chọn một cấu trúc hai chiều để biểu diễn lời giải (một nhiễm sắc thể) của bài toán vận tải: một ma trận $V= (x_{ij})$ $(1 \le i \le k, 1 \le j \le n)$. Lần này mỗi x_{ij} là một số thực.

7.2.2. Khởi tạo

Thủ tục khởi tạo cũng giống như trong trường hợp tuyến tính (phần 7.1.3). Cũng vậy, nó tạo một ma trận có tối đa $k+n-1$ các phần tử khác 0 sao cho các tất cả các ràng buộc đều được thỏa.

7.2.3. Lượng giá

Trong trường hợp này ta phải tối ưu hóa chi phí, một hàm phi tuyến của các giá trị của ma trận.

7.2.4. Các toán tử

Ta định nghĩa hai toán tử di truyền, *đột biến* và *lai số học*.

ĐỘT BIẾN

Có hai loại đột biến được định nghĩa. Loại thứ nhất, **đột_biến-1,** giống như toán tử đã được dùng trong trường hợp tuyến tính và đưa vào càng nhiều số 0 vào ma trận càng tốt. Loại thứ hai, **đột_biến-2,** được hiệu chỉnh để tránh chọn phải các số 0 bằng cách chọn các giá trị trong một khoảng. Toán tử **đột_biến-2** cơ bản cũng như **đột_biến-1** ngoại trừ việc sử dụng một phiên bản được hiệu chỉnh của thủ tục **khởi_tạo** để tính toán lại các ma trận con được chọn.

Trong thủ tục **khởi_tạo** ở phần 7.1.3 ta thay dòng:

val ← min (sour[i], dest [j])

bằng các lệnh:

val ← min (sour[i], dest [j])

Nếu (*i* là hàng sẵn có cuối cùng) Or (*j* là cột sẵn có cuối cùng) **Thì:**

val ← val$_1$

Ngược lại *val* ← số thực ngẫu nhiên trong khoảng <0, *val*$_1$>

Thay đổi này tạo ra các số thực thay vì các số nguyên hoặc số 0, nhưng thủ tục này cần được biến đổi thêm vì hiện tại nó tạo ra ma trận có thể vi phạm các ràng buộc.

Thí dụ, sử dụng lại ma trận ở Thí dụ 7.1, giả sử chuỗi các số được chọn là (3, 6, 12, 8, 10, 1, 2, 4, 9, 11, 7, 5) và số thực đầu tiên được phát sinh cho số 3 (hàng 1 cột 3) là 7.3 (là số thuộc khoảng

[0.0, min(sour[1], dest[3])] = [0.0, 15.0]). Số thực ngẫu nhiên thứ hai cho số 6 (hàng 2, cột 2) là 12.1, và các số thực còn lại được phát sinh bằng thủ tục **khởi_tạo** mới là: 3.3, 5.0, 1.0, 3.0, 1.9, 1.7, 0.4, 0.3, 7.4, 0.5. Ma trận kết quả là:

	5.0	**15.0**	**15.0**	**10.0**
15.0	3.0	1.9	7.3	1.7
25.0	0.5	12.1	7.4	5.0
5.0	0.4	1.0	0.3	3.3

Chỉ cần cộng thêm 1.1 vào phần tử x_{11} các ràng buộc có thể được thỏa mãn hoàn toàn. Như vậy, ta cần thêm vào dòng cuối cùng của thuật giải **đột_biến-2** câu:

"Tạo các điều kiện cần thiết"

Điều này bổ sung cho việc thay đổi thủ tục **khởi_tạo.**

LAI TẠO

Bắt đầu với hai cha-mẹ (các ma trận *U* và *V*) toán tử lai sẽ sinh ra hai con *X* và *Y*, trong đó:

$X = c_1*U + c_2*V$ và:

$Y = c_1*V + c_2*U,$

(với c_1, $c_2 \geq 0$ *và* $c_1+c_2 = 1$). Do tập ràng buộc buộc là lồi nên việc thực hiện này bảo đảm rằng cả hai con sinh ra đều thỏa mãn nếu cả cha và mẹ chúng thỏa mãn ràng buộc. Đây là cách giản đơn hóa rất quan trọng của trường hợp tuyến tính mà ở đây có thêm một yêu cầu nữa là giữ lại tất cả các thành phần của ma trận là các số nguyên.

7.2.5. Các tham số

Ngoài việc thiết lập các tham số điều khiển như đã sử dụng trong trường hợp bài toán tuyến tính (kích thước quần thể, xác suất đột biến và lai tạo, hạt giống ban đầu (số ngẫu nhiên), vv...) còn cần thêm một số tham số khác nữa. Đó là những hệ số lai tạo, c_1 và c_2, và m_1, một tham số xác định tỉ lệ để **đột_biến-1** được thực hiện trong các lần xảy ra đột biến.

7.2.6. Các ví dụ thực nghiệm

Để có thể hình dung được hiệu quả của phương pháp được đề nghị, chúng tôi chọn minh họa phương pháp với bài toán vận tải 7×7 (bảng 7.2) và thử nghiệm với nhiều hàm mục tiêu khác nhau.

	20	20	20	23	26	25	26
27	x_1	x_2	x_3	x_4	x_5	x_6	x_7
28	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
25	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}
20	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}
20	x_{29}	x_{30}	x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
20	x_{36}	x_{37}	x_{38}	x_{39}	x_{40}	x_{41}	x_{42}
20	x_{43}	x_{44}	x_{45}	x_{46}	x_{47}	x_{48}	x_{49}

Bảng 7.2. Bài toán vận tải 7×7.

Bài toán là:

$$Min\ f(\mathbf{x}) = f(x_1,\dots,x_{49}),$$

Thỏa 14 phương trình (với 13 phương trình độc lập) sau:

$$x_1+x_2+x_3+x_4+x_5+x_6+x_7 = 27$$

$$x_8+x_9+x_{10}+x_{11}+x_{12}+x_{13}+x_{14} = 28$$

$$x_{15}+x_{16}+x_{17}+x_{18}+x_{19}+x_{20}+x_{21} = 25$$

$$x_{22}+x_{23}+x_{24}+x_{25}+x_{26}+x_{27}+x_{28} = 20$$

$$x_{29}+x_{30}+x_{31}+x_{32}+x_{33}+x_{34}+x_{35} = 20$$

$$x_{36}+x_{37}+x_{38}+x_{39}+x_{40}+x_{41}+x_{42} = 20$$

$$x_{43}+x_{44}+x_{45}+x_{46}+x_{47}+x_{48}+x_{49} = 20$$

$$x_1+x_8+x_{15}+x_{22}+x_{29}+x_{36}+x_{48} = 20$$

$$x_2+x_9+x_{16}+x_{23}+x_{30}+x_{37}+x_{44} = 20$$

$$x_3+x_{10}+x_{17}+x_{24}+x_{31}+x_{38}+x_{45} = 20$$

$$x_4+x_{11}+x_{18}+x_{25}+x_{32}+x_{39}+x_{46} = 23$$

$$x_5+x_{12}+x_{19}+x_{26}+x_{33}+x_{40}+x_{47} = 26$$

$$x_6+x_{13}+x_{20}+x_{27}+x_{34}+x_{41}+x_{48} = 25$$

$$x_7+x_{14}+x_{21}+x_{28}+x_{35}+x_{42}+x_{49} = 26$$

Cách ứng xử của các thuật giải tối ưu phi tuyến tùy thuộc chặt chẽ vào dạng của hàm mục tiêu. Những kỹ thuật giải khác nhau rõ ràng sẽ có những đáp ứng khác nhau.



Vì mục đích thử nghiệm, ta đã chọn một lớp ngẫu nhiên các hàm mục tiêu mạnh thành các hàm sẽ gặp trong các bài toán OR (hàm thực hành), những bài toán này thường được minh hoạ trong các sách về tối ưu (hàm hợp lý) và những bài toán thường gặp hơn trong các trường hợp thử nghiệm khó của các kỹ thuật tối ưu hóa (hàm khác). Tóm lại, chúng có thể được mô tả như sau:

- Hàm thực hành

Các hàm chi phí tuyến tính từng khúc thường xuất hiện trong thực hành vì những hạn chế về mặt dữ liệu hay vì những thực hiện phương tiện với những lãnh vực áp dụng những chi phí khác nhau, chúng thường là hàm không trơn và vì thế đạo hàm chắc chắn không liên tục. Chúng thường gây khó khăn cho những phương pháp dựa trên gradient mặc dù có thể chuyển chúng thành các hàm khả vi. Thí dụ : A(x) và B(x).

- Hàm hợp lý

Đây là những hàm trơn.

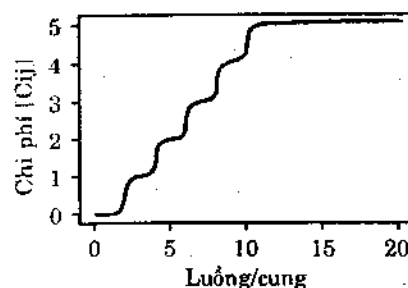
- Các hàm khác

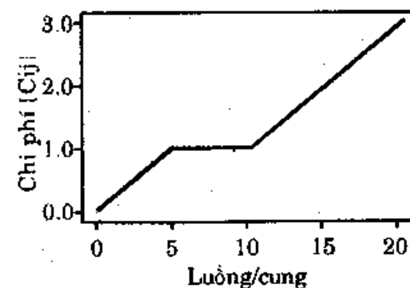
Những hàm này tiêu biểu có nhiều vùng trũng (hoặc nhiều đỉnh) với nhiều tối ưu cục bộ gây rất nhiều khó khăn cho các phương pháp gradient. Chúng được dùng cho những thử nghiệm nghiêm khắc để đánh giá các thuật giải tối ưu và ta hy vọng chúng ít xuất hiện trong thực tế. Thí dụ E(x) và F(x).

Dưới đây hai thí dụ điển hình, từ mỗi nhóm hàm mục tiêu sử dụng trong các thử nghiệm. Chúng là những hàm riêng biệt của những thành phần thuộc vectơ lời giải không có các số hạng lai. Những phiên bản liên tục hóa của các đồ thị được trình bày trong hình 7.3.

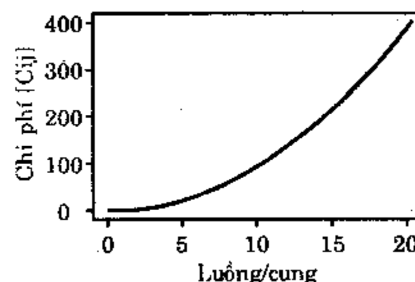


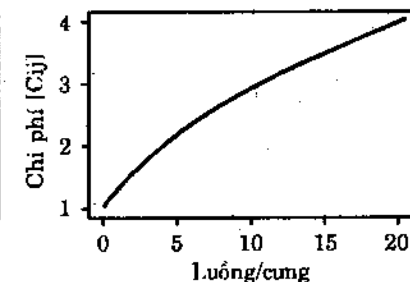
Chi phí cung của hàm mục tiêu A (tham số =10) Chi phí cung của hàm mục tiêu B (tham số =5)



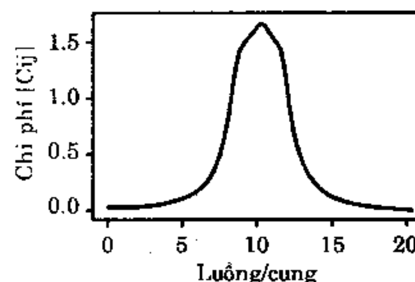


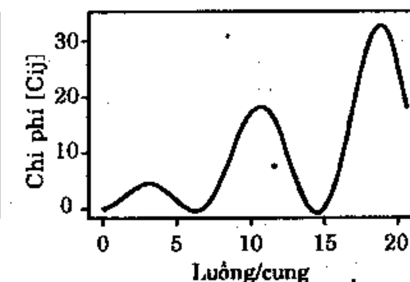
Chi phí cung của hàm mục tiêu   Chi phí cung của hàm mục tiêu D (tham số =1)





Chi phí cung của hàm mục tiêu E   Chi phí cung của hàm mục tiêu F





Hình 7.3. Sáu hàm thử nghiệm A - F



$$\text{- Hàm } A(x) = \begin{cases} 0, & \text{nếu} \quad 0 < x \le S \\ c_{ij}, & \text{nếu} \quad S < x \le 2S \\ 2c_{ij}, & \text{nếu} \quad 2S < x \le 3S \\ 3c_{ij}, & \text{nếu} \quad 3S < x \le 4S \\ 4c_{ij}, & \text{nếu} \quad 4S < x \le 5S \\ 5c_{ij}, & \text{nếu} \quad 5S < x \end{cases}$$

trong đó, S nhỏ hơn giá trị x cụ thể.

$$\text{- Hàm } B(x) = \begin{cases} c_{ij}\frac{x}{S}, & \text{nếu} \quad 0 < x \le S \\ c_{ij}, & \text{nếu} \quad S < x \le 2S \\ c_{ij}(1 + \frac{x-2S}{S}), & \text{nếu} \quad 2S < x \end{cases}$$

trong đó, S là bậc của giá trị x cụ thể.

- Hàm $C(x) = c_{ij}x^2$

- Hàm $D(x) = c_{ij}\sqrt{x}$

- Hàm $E(x) = c_{ij}\left(\frac{1}{1+(x-2S)^2} + \frac{1}{1+(x-\frac{9}{4}S)^2} + \frac{1}{1+(x-\frac{7}{4}S)^2}\right)$

trong đó, S là bậc của giá trị x cụ thể.

- Hàm $F(x) = c_{ij}x(\sin(x\frac{5\pi}{4S}) + 1)$

Trong đó, S là bậc của giá trị x cụ thể.

Hàm mục tiêu của bài toán vận tải thuộc dạng $\sum_{ij} f(x_{ij})$



trong đó, $f(x)$ là một trong 6 hàm trên, các tham số c_{ij} nhận được từ ma trận tham số (xem hình 7.4), và S nhận được từ các bài toán được thử nghiệm.

Để tính đạo hàm S cần lượng giá một giá trị x đặc biệt; điều này được thực hiện bởi các lần chạy đầu tiên để lượng giá số lượng và tầm quan trọng của các x_{ij} khác 0. Theo cách này, ta tính được luồng trung bình trên mỗi cung và tìm ra một giá trị của S. Đối với hàm A ta dùng $S = 2$ và dùng $S = 5$ cho B, E, F.

Chú ý rằng hàm mục tiêu trên mỗi cung đều giống nhau, vì vậy một ma trận chi phí được sử dụng để cho biết thay đổi giữa các cung. Ma trận dạng c_{ij} có nhiệm vụ định tỉ lệ dạng hàm cơ bản, như vậy cung cấp 'một độ' thay đổi.

7.2.7. Thử nghiệm và kết quả

Khi thử nghiệm thuật giải GENETIC-2 trên bài toán vận tải tuyến tính (phần 7.1) ta có thể so sánh lời giải của nó với lời giải tối ưu đã biết tìm được bằng các thuật giải chuẩn (như phương pháp đơn hình chẳng hạn). Như vậy ta có thể xác định hiệu quả của thuật giải di truyền. Khi ta thực nghiệm với các hàm mục tiêu phi tuyến, có thể không biết được tối ưu chính xác. Việc thử nghiệm chỉ còn là so sánh các kết quả với kết quả của những phương pháp lời giải phi tuyến khác mà có thể chính chúng đã hội tụ vào một tối ưu cục bộ.

Số nguồn: 7

Số đích : 7

Các luồng nguồn: 27, 28, 25, 20, 20, 20, 20

Các luồng đích: 20, 20, 20, 23, 26, 25, 26

Mạ trận tham số cung (Nguồn theo đích)

0	21	50	62	93	77	1000
21	0	17	54	67	1000	48
50	17	0	60	98	67	25
62	54	60	0	27	1000	38
93	67	98	27	0	47	42
77	1000	67	1000	47	0	35
1000	48	25	38	42	35	0

Hình 7.4. Mô tả bài toán thí dụ.

Như thường lệ, ta so sánh phương pháp thuật giải GENETIC-2 với hệ GAMS tiêu biểu của phương pháp chuẩn. Hệ GAMS này, chính là cài đặt phương pháp gradient, có thể khó hoặc không giải nổi một số bài toán mà ta thử nghiệm. Trong những trường hợp này, có thể tạo một số hiệu chỉnh cho các hàm mục tiêu để ít nhất phương pháp này cũng tìm được một lời giải gần đúng.

Lúc này, mục tiêu của bài toán vận tải có dạng

$$\sum_{ij} f(x_{ij})$$

trong đó, *f(x)* là một trong 6 hàm trên được chọn, các tham số c_{ij} nhận được từ ma trận tham số và *S* là từ các thuộc tính của bài toán được thử nghiệm. *S* được tính gần đúng từ luồng cung trung bình không là 0, được xác định từ một số lần chạy lúc đầu để bảo đảm các luồng xuất hiện trong phần thích hợp của hàm mục tiêu.

Ở một khía cạnh nào đó, các hàm mục tiêu được cấu trúc ngẫu nhiên hoàn toàn thích hợp để dùng trên mỗi cung. Hàm mục tiêu của ta được cho là để minh họa thuật giải đối với nhiều loại bài toán và câu hỏi rút xuống là có bao nhiêu biến đổi cần thiết giữa các cung cho một dạng hàm cụ thể. Khi hàm giống hệt nhau trên mỗi cung, bài toán có thể có nhiều lời giải với cùng chi phí, giảm thộng tin nhận được khi phân tích thuật giải.

Trong các thử nghiệm của ta, một ma trận - chi phí được sử dụng để cung cấp biến đổi giữa các cung. Ma trận cung cấp các c_{ij} có nhiệm vụ định tỉ lệ dạng hàm cơ bản, như vậy cung cấp 'một độ' thay đổi. Thêm ma trận (cung cấp nhiều độ thay đổi) là không cần thiết.

Đối với các hàm C, E, và F có thể áp dụng GAMS: dùng các hàm phi tuyến cài sẵn. Do yêu cầu lượng giá gradient của các hàm mục tiêu, GAMS không thể xử lý các hàm A, B, và D một cách trực tiếp. Trong trường hợp của A và B, không thể xây dựng công thức cho biểu thức trong GAMS, trong khi trường hợp của D (hàm rút căn bậc hai) lại có khó khăn trong việc đo các gradient gần 0. Vì vậy, có những hiệu chỉnh sau đây cho bài toán đối với các lần chạy GAMS,

- Hàm A

Các hàm arctan riêng biệt được sử dụng để tính gần đúng mỗi bước trong 5 bước sau đây. Tham số A, P_A được dùng để điều khiển độ 'khít khảo' của thích nghi. Chi phí trên cung [*i, j*] là:

$$c_{ij} \cdot \left(\begin{array}{l} \arctan(P_A(x_{ij} - S))/\pi + \frac{1}{2} + \\ \arctan(P_A(x_{ij} - 2S))/\pi + \frac{1}{2} + \\ \arctan(P_A(x_{ij} - 3S))/\pi + \frac{1}{2} + \\ \arctan(P_A(x_{ij} - 4S))/\pi + \frac{1}{2} + \\ \arctan(P_A(x_{ij} - 5S))/\pi + \frac{1}{2} \end{array} \right)$$

- Hàm B

Hàm arctan lại được sử dụng, lần này để tính gần đúng ba gradient. Một tham số P_b được sử dụng để điều khiển độ khít khao của thích nghi. Chi phí trên cung [i,j] là:

$$c_{ij} \cdot \left(\begin{array}{l} (\frac{x_{ij}}{S}).(\arctan(P_B x_{ij})/\pi + \frac{1}{2}) + \\ (1 - \frac{x_{ij}}{S}).(\arctan(P_B x_{ij})/\pi + \frac{1}{2}) + \\ (\frac{x_{ij}}{S} - 2).(\arctan(P_B x_{ij})/\pi + \frac{1}{2}) + \end{array} \right)$$

- Hàm D

Để tránh những vấn đề về gradient tại/gần 0, hàm D được đổi thành:

$D'(x) = D(x + \varepsilon)$

Cũng như trong trường hợp tuyến tính, đối với những bài toán nhiều lần chạy GAMS được tạo với các giá trị khác nhau của tham số hiệu chỉnh và kết quả tốt nhất được chọn. Các giá trị tốt nhất cho ba tham số tìm được là: P_A giữa 1 và 20, P_B rất lớn (thí dụ 1000), và ε (cho hàm D) giữa 1 và 7. Các giá trị kết quả cuối cùng luôn luôn được tính sau khi tối ưu hóa bằng hàm không được hiệu chỉnh, thay vì hàm hiệu chỉnh.

Đối với tập thử nghiệm chính, 5 ma trận vận tải 10×10 được dùng cho mỗi hàm. Chúng được xây dựng từ một tập các giá trị độc lập được phân phối đều c_{ij} và các vectơ nguồn và đích được chọn ngẫu nhiên với toàn luồng gồm 100 đơn vị. Mỗi tổ hợp hàm - ma trận chạy với 5 lần, sử dụng các hạt giống khởi đầu là số ngẫu nhiên cho thuật giải di truyền. Một lần chạy 10000 thế hệ.

S được cho giá trị bằng 2 đối với hàm A, nhưng bằng 5 cho các hàm B, E và F.

Dùng những bài toán lớn hơn nhiều để so sánh hệ thống di truyền với những phương pháp giải theo kiểu qui hoạch phi tuyến có lẽ chỉ có giá trị hạn chế. Kết quả của 10 × 10 lần chạy cho thấy khuynh hướng rơi vào tối ưu cục bộ (không toàn cục) của GAMS (và có thể nói, cả những hệ thống tương tự). Bỏ qua thời gian dùng để lượng giá hàm mục tiêu và sử dụng số lời giải được thử nghiệm làm mức đo thời gian thì cũng thấy rõ là những kỹ thuật qui hoạch phi tuyến chuẩn sẽ luôn 'chấm dứt' nhanh hơn các hệ thống tiến hóa. Điều này là do chúng chỉ khai thác tiêu biểu một đường dẫn đặc biệt bên trong vùng tối ưu cục bộ hiện hành. Chúng sẽ làm tốt chỉ khi tối ưu cục bộ tương đối tốt.

Một tập các tham số được chọn cho GENETIC-2 sau khi đã có kinh nghiệm với các bài toán tuyến tính và trên cơ sở những lần chạy thử với các bài toán phi tuyến. Kích thước quần thể giữ nguyên là 40. Tỉ lệ đột biến là p_m = 20% với tỉ lệ **đột_biến-1** là 50% và tỉ lệ lai p_c = 55%. Các tỉ lệ lai là c_1 = .35 và c_2 =.65.

Có vẻ là tỉ lệ đột biến được chọn quá cao và tỉ lệ lai quá thấp so với các thuật giải di truyền cổ điển. Nhưng các toán tử lại khác với các toán tử cổ điển, vì (1) ta chọn cha-mẹ cho đột biến và lai, nghĩa là, *toàn bộ* cấu trúc (trái với các bit đơn) trải qua đột biến, và (2) **đột_biến-1** tạo một con 'đẩy' cha-mẹ đến bề mặt của không gian lời

giải, trong khi lai và **đột_biến-2** 'đẩy' con vào tâm của không gian lời giải.

Việc sử dụng các tỉ lệ đột biến cao cũng có thể gợi ý rằng thuật giải gần như là tìm kiếm ngẫu nhiên. Nhưng, thuật giải tìm kiếm ngẫu nhiên (tỉ lệ lai tạo 0%) thực hiện rất tồi so với thuật giải đã sử dụng ở đây. Để chứng minh điều này, ta lập bảng dưới đây (bảng 7.3) một số kết quả tiêu biểu cho những giá trị khác nhau của các tham số p_m và p_c bằng các hàm A và F và cho bài toán vận tải 7×7 cụ thể. Các giá trị được cho là chi phí tối thiểu trung bình nhận được trong 5 lần chạy với các hạt giống khác nhau trong 10000 thế hệ.

Hàm	$p_c = 0\%$ $p_m = 25\%$	$p_c = 25\%$ $p_m = 0\%$	$p_c = 5\%$ $p_m = 20\%$
A	45.8	181.0	0.0
F	187.7	189.6	110.9

Bảng 7.3. các kết quả cho những giá trị khác nhau của p_c và p_m .

Bài toán vận tải 7×7 đã dùng được cho trong hình 7.4; các lời giải tìm được bằng thuật giải GENETIC-2 đối với các giá trị khác của các tham số p_c và p_m có trong hình 7.5.

$p_c = 0\%$, $p_m = 25\%$, hàm A, chi phí 45.8

20.00	0.00	0.00	1.00	2.00	2.00	2.00
0.00	20.00	1.00	2.00	2.00	2.00	1.00
0.00	0.00	19.00	0.00	2.00	1.00	3.00
0.00	0.00	0.00	20.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	20.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	20.00

$p_c = 0\%$, $p_m = 25\%$, hàm F, chi phí 178.7

15.00	6.00	0.00	6.00	0.00	0.00	0.00
0.00	14.00	0.00	14.00	0.00	0.00	0.00
0.00	0.00	20.00	0.00	0.00	0.00	5.00
5.00	0.00	0.00	3.00	6.00	0.00	6.00
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	20.00	0.00
0.00	0.00	0.00	0.00	0.00	5.00	15.00

$p_c = 0\%$, $p_m = 25\%$, hàm A, chi phí 181.2

18.25	0.00	0.11	1.81	3.30	3.53	0.00
0.00	18.21	3.94	3.05	0.00	1.97	0.82
1.75	1.79	13.74	1.46	1.46	1.48	3.83
0.00	0.00	1.91	14.37	1.18	0.35	1.69
0.00	0.00	0.74	0.97	18.10	0.21	0.00
0.00	0.00	0.52	0.71	1.96	17.30	0.00
0.00	0.00	0.05	0.14	0.00	0.16	19.65

$p_c = 25\%$, $p_m = 0\%$, hàm F, chi phí 189.8

20.00	0.25	0.00	0.75	0.00	6.00	0.00
0.00	5.75	0.00	12.25	0.00	0.00	0.00
0.00	0.00	20.00	0.00	0.00	0.00	5.00
0.00	14.00	0.00	0.00	6.00	0.00	0.00
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	14.00	6.00
0.00	0.00	0.00	0.00	0.00	5.00	15.00

$p_c = 5\%$, $p_m = 20\%$, hàm A, chi phí 0.0

19.87	0.00	0.68	1.80	1.33	1.60	1.51
0.08	20.00	1.00	1.90	1.48	1.61	1.92
0.00	0.00	18.32	1.89	1.08	1.78	1.93
0.05	0.00	0.00	17.09	1.91	0.96	0.00
0.00	0.00	0.00	0.00	19.52	0.00	0.08
0.00	0.00	0.00	0.00	0.00	18.60	1.40
0.00	0.00	0.00	0.31	0.25	0.25	19.16

$p_c = 5\%$, $p_m = 20\%$, hàm F, chi phí 110.9

14.31	6.31	6.39	0.00	0.00	0.00	0.00
0.00	13.69	0.31	14.00	0.00	0.00	0.00
0.00	0.00	13.31	6.00	0.00	0.00	5.69
5.69	0.00	0.00	3.00	6.00	0.00	5.31
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	19.31	0.69
0.00	0.00	0.00	0.00	0.00	5.69	14.31

Hình 7.4. Các lời giải tìm được bằng thuật giải GENETIC-2 đối với các giá trị khác của các tham số p_c và p_m.

Một so sánh tiêu biểu về tối ưu giữa GENETIC-2 (tính trung bình trên 5 hạt giống) và GAMS được trình bày trong bảng 7.5 của một bài toán 10×10; Hình 7.6 mô tả điều này.

Hình 7.7 trình bày các kết quả của cả 5 bài toán được xét. Đối với lớp các bài toán 'thực hành', A và B, trung bình thì GENETIC-2 tốt hơn GAMS 24.5% trong trường hợp A và 11.5% trong trường hợp B. Đối với các hàm 'hợp lý' các kết quả lại khác. Trong trường hợp C (hàm bình phương) hệ thống di truyền thực hiện dở hơn 7.5% trong khi trong trường hợp D (hàm căn bậc hai), hệ thống di truyền trung bình tốt hơn 2.0%. Đối với những hàm 'khác', E và F, hệ thống di truyền thống trị; nó đưa đến những cải thiện 33.0% và 54.5% hơn hẳn GAMS tính trung bình trên 5 bài toán.

7.2.8. Kết luận

Mục đích của ta là khám phá các phản ứng của một loại thuật giải di truyền đối với những bài toán nhiều ràng buộc và nhiều hàm mục tiêu phi tuyến. Bài toán vận tải được chọn để nghiên cứu vì nó

cung cấp một tập lời giải lồi, tương đối đơn giản. Điều này giúp dễ dàng hơn trong việc duy trì tính thỏa mãn của các lời giải. Như vậy ta có thể khảo sát tác động chỉ của hàm mục tiêu trên các phản ứng của thuật giải.

Hàm	GAMS	GENETIC -2	% sai biệt
A	281.0	202.0	-28.1
B	180.8	163.0	-9.8
C	4402.0	4556.2	+3.5
D	408.4	391.1	-4.2
E	145.1	79.2	-45.4
F	1200.8	201.9	-83.2

Bảng 7.5. So sánh giữa GAMS và GENETIC-2.

Số nguồn: 10

Số đích: 10

Các luồng nguồn: 8, 8, 2, 26, 12, 1, 6, 18, 18, 1,

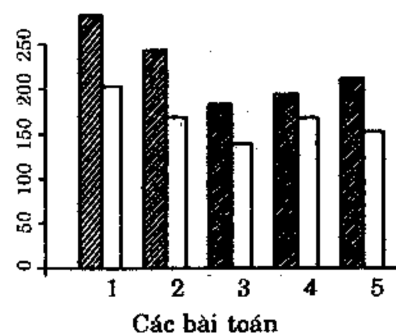
Các luồng đích: 19, 2, 33, 5, 11, 11, 2, 14, 2, 1

Ma trận tham số cung (nguồn theo đích)

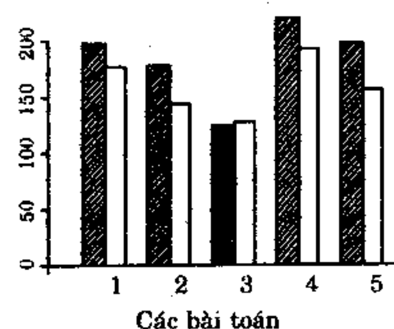
15	3	23	1	19	14	6	16	41	33
13	17	30	36	20	17	26	19	3	33
37	17	30	5	48	27	8	25	36	21
13	13	31	7	35	11	20	41	34	3
31	24	8	30	28	33	2	8	1	8
32	36	12	9	18	1	44	49	11	11
49	6	17	0	42	45	22	9	10	47
2	21	18	40	47	27	27	49	19	42
13	16	25	21	19	0	32	20	32	35
23	42	2	0	9	30	5	29	31	29

Hình 7.6. Mô tả bài toán thí dụ.

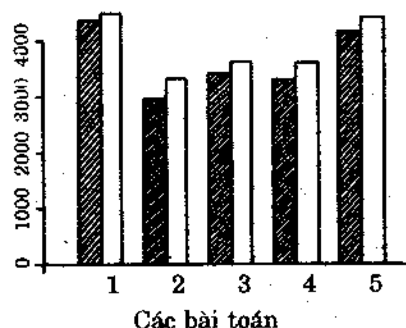
Các chi phí tìm được qua hàm A



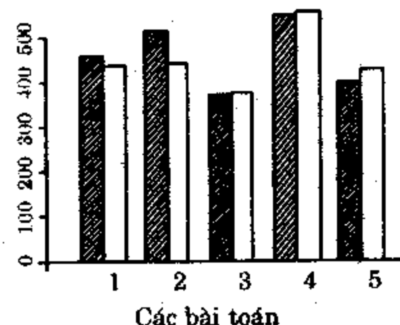
Các chi phí tìm được qua hàm B



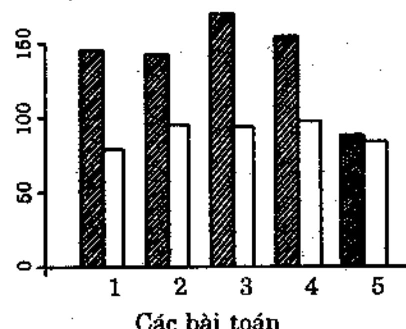
Các chi phí tìm được qua hàm C



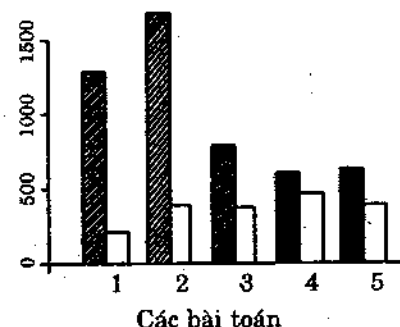
Các chi phí tìm được qua hàm D



Các chi phí tìm được qua hàm E



Các chi phí tìm được qua hàm F



Hình 7.7. Các kết quả. Gạch chéo: GAMS. Trơn: GENETIC-2 trung bình

Các kết quả trên cho thấy hiệu quả của phương pháp di truyền trong việc tìm kiếm tối ưu toàn cục của những bài toán khó, mặc dù GAMS thực hiện tốt trên các hàm trơn, đơn điệu, 'hợp lý'. Các kỹ thuật được điều khiển bởi gradient thích hợp nhất cho các tình huống này. Đối với hàm C, GAMS tìm được những lời giải tốt hơn nhanh hơn GENETIC-2 nhiều.

Với các bài toán 'thực hành', các kỹ thuật gradient gặp khó khăn 'chỉ thấy gần' các vùng mới của các chi phí tốt hơn. Thuật giải

di truyền, với cách tiếp cận toàn cục hơn, có thể sẵn sàng chuyển đến những vùng mới do đó phát sinh nhiều lời giải tốt hơn nhiều.

- Những bài toán 'khác', mặc dù chúng trơn, nhưng có các tính năng cấu trúc quan trọng được thiết kế gây khó khăn thực sự cho những phương pháp gradient. GENETIC-2 ở đây xuất sắc hơn GAMS, thậm chí còn nhiều hơn trong các trường hợp 'thực hành'.

Việc so sánh GENOCOP (chương 6) với GENETIC-2 cũng thú vị (xem bảng 7.5). Nói chung, kết quả của chúng rất giống nhau nhưng lại cần chú ý rằng, phương pháp ma trận được thiết kế theo bài toán đặc tả (vận tải), trong khi GENOCOP độc lập với bài toán và hoạt động không có một tri thức miền được mã hóa cứng. Nói cách khác, trong khi ta có thể mong đợi GENOCOP thực hiện cũng tốt như thế đối với các bài toán ràng buộc khác, thì GENETIC-2 không thể nào sử dụng được.

Hàm	GENETIC II	GENOCOP	% sai biệt
A	00.00	24.15	
B	203.81	205.60	0.87
C	2564.23	2571.04	0.26
D	480.16	480.16	0.00
E	204.73	204.82	0.04
F	110.94	119.61	7.24

Bảng 7.5. GENETIC-2 đối với GENOCOP : các kết quả của bài toán 7x7 với các hàm chi phí vận tải A-F và ma trận chi phí đã cho trong hình 7.3

Khi so sánh cả 3 hệ thống (GAMS, GENOCOP, GENETIC-2) cần lưu ý rằng hai hệ thống GAMS và GENOCOP độc lập bài toán: chúng có khả năng tối ưu hóa bất cứ hàm nào có liên quan đến tập các ràng buộc tuyến tính. Hệ thống thứ ba GENETIC-2 chỉ được thiết kế cho các bài toán vận tải: các ràng buộc đặc biệt được kết hợp vào các cấu trúc dữ liệu ma trận và các toán tử "di truyền" đặc biệt.

GENETIC-2 được thiết kế đặc biệt cho các bài toán vận tải nhưng có một đặc trưng quan trọng là nó xử lý bất cứ loại hàm chi phí nào (không cần phải liên tục). Cũng có thể hiệu chỉnh nó để xử lý nhiều bài toán nghiên cứu những toán tử giống nhau như một số bài toán lập thời khóa biểu. Đây có vẻ là một hướng nghiên cứu hứa hẹn đưa đến một kỹ thuật chung để giải những bài toán tối ưu hóa có ràng buộc dựa trên ma trận.

Phần 3

TỐI ƯU TỔ HỢP

Chương 8

BÀI TOÁN NGƯỜI DU LỊCH

Trong các chương tiếp theo, chúng tôi sẽ trình bày các chương trình tiến hóa được thiết kế cho những ứng dụng đặc hiệu (về đồ thị, phân hoạch, lập lịch). Bài toán người du lịch (TSP – Traveling Salesman Problem) chỉ là một trong những ứng dụng đó; nhưng ta coi nó là một bài toán đặc biệt và dành hẳn một chương để bàn về nó. Lý do vì sao?

Có rất nhiều lý do. Trước tiên, về ý niệm, TSP rất đơn giản: một du khách phải ghé mỗi thành phố trong vùng của anh ta, chính xác một lần, rồi trở về điểm khởi hành. Chi phí du lịch giữa từng cặp thành phố được cho trước, anh ta phải lập kế hoạch cho hành trình của mình ra sao để tổng chi phí cho toàn hành trình là tối thiểu? Không gian tìm kiếm của TSP là tập các hoán vị của n thành phố. Bất cứ một hoán vị nào của n thành phố này cũng là một lời giải chấp nhận được (là toàn bộ *một* hành trình qua các thành phố). Lời giải tối ưu là một hoán vị với chi phí tối thiểu của hành trình. Kích thước của không gian tìm kiếm là $n!$.

TSP là một bài toán tương đối cổ điển: có tài liệu minh chứng bài toán này đã xuất hiện từ năm 1759 và bởi Euler, người có hứng thú giải bài toán mã đi tuần. Lời giải là chuỗi 64 ô cờ mà quân mã phải đi qua, mỗi ô đúng một lần.

TSP được tập đoàn RAND giới thiệu vào 1948. Danh tiếng của tập đoàn này giúp TSP trở thành bài toán phổ biến và nổi tiếng. Vào lúc đó, TSP cũng phổ biến do qui hoạch tuyến tính là một vấn đề mới và do có nhiều nghiên cứu tập trung giải những bài toán tổ hợp.

Bài toán TSP được chứng minh thuộc loại NP–khó. Nó xuất hiện trong nhiều ứng dụng và số thành phố là hoàn toàn có ý nghĩa.

Trong những thập kỷ gần đây, đã xuất hiện nhiều thuật giải đạt gần đến được lời giải tối ưu của bài toán TSP: thuật giải láng giềng gần nhất, thuật giải háu ăn, đảo gần nhất, đảo xa nhất, các thuật giải do Karp, Like, Christofides, vv... Một nhóm thuật giải khác (2–opt, 3–opt, Lin–Kernighan) nhắm vào tối ưu cục bộ: cải thiện hành trình bằng các rối loạn cục bộ. TSP cũng trở thành một đích nhắm của cộng đồng GA: nhiều thuật giải dựa trên di truyền đã được báo cáo trong những năm gần đây. Những thuật giải này nhằm tạo ra những lời giải gần tối ưu, bằng cách duy trì một quần thể các lời giải trải qua những biến đổi một ngôi và hai ngôi ('đột biến' và 'lai tạo') theo một kế hoạch chọn lọc nhiều về phía những cá thể phù hợp. Rất hay khi so sánh những phương pháp này, đặc biệt cần quan tâm đến việc biểu diễn và những toán tử di truyền được sử dụng – đây là những gì ta muốn thực hiện trong chương này. Nói cách khác, ta sẽ lần theo sự tiến hóa của những chương trình tiến hóa đối với TSP.

Để làm nổi bật một số đặc trưng quan trọng của TSP, trước tiên ta hãy xét bài toán CNF–có thể thỏa. Một biểu thức lôgic ở dạng nối liền CNF là một chuỗi các mệnh đề liên kết với nhau theo phép $\wedge$ (and); một mệnh đề là một chuỗi các literal liên kết với nhau bởi toán tử $\vee$ (or), một literal là một biến lôgic hay là phủ định của nó; biến lôgic là biến có thể có giá trị TRUE hoặc FALSE (1 hay 0).

Thí dụ, biểu thức lôgic sau đây có dạng CNF:

$$(a \vee \bar{b} \vee c) \wedge (b \vee c \vee d \vee \bar{e}) \wedge (\bar{a} \vee c) \wedge (a \vee \bar{c} \vee \bar{e})$$

trong đó, a, b, c, d, e là các biến lôgic; $\bar{a}$ là phủ định của biến a ($\bar{a}$ có giá trị TRUE nếu và chỉ nếu a có giá trị FALSE).

Vấn đề là tìm giá trị các biến trong biểu thức sao cho biểu thức có giá trị là TRUE. Thí dụ, biểu thức lôgic CNF trên có nhiều nghiệm với a = TRUE và c = TRUE, các biến khác có giá trị bất kỳ.

Nếu ta thử áp dụng một thuật giải di truyền vào bài toán CNF– có thể thỏa, ta nhận thấy rằng khó tưởng tượng bài toán có một biểu diễn khác phù hợp hơn: một vectơ nhị phân chiều dài cố định (chiều dài của vectơ tương ứng với số biến) sẽ đảm trách công việc. Hơn nữa, không có phụ thuộc nào giữa các bit: thay đổi nào cũng đưa đến một lời giải (có nghĩa) hợp lệ. Như vậy ta có thể áp dụng đột biến và lai tạo mà không cần các bộ giải mã hay các thuật giải sửa chữa. Tuy vậy, việc chọn hàm lượng giá là công việc khó khăn nhất. Chú ý rằng tất cả các biểu thức lôgic được đánh giá TRUE hay FALSE và nếu có một phép gán giá trị cụ thể đánh giá cả biểu thức là TRUE, thì đã tìm được lời giải của bài toán. Vấn đề là trong khi tìm kiếm lời giải, tất cả các nhiễm sắc thể (các vectơ) trong một quần thể có thể có giá trị là FALSE (trừ phi kiếm được lời giải), vì thế không thể phân biệt giữa các nhiễm sắc thể 'tốt' và 'xấu'. Tóm lại, bài toán CNF– có thể thỏa có biểu diễn và các toán tử tự nhiên, mà không có hàm lượng giá tự nhiên.

Mặt khác, TSP có một hàm lượng giá cực dễ (và tự nhiên): để có một lời giải (hoán vị các thành phố) ta có thể tham chiếu bảng khoảng cách giữa tất cả các thành phố và (sau $n-1$ toán tử bổ sung) ta có toàn bộ chiều dài của hành trình. Thế nhưng, trong quần thể các hành trình, ta dễ dàng so sánh một cặp bất kỳ trong chúng. Tuy nhiên, việc chọn biểu diễn của một hành trình và việc chọn các toán tử cần dùng đều không có gì rõ ràng.

Một lý do khác nữa để nói về TSP trong một chương riêng biệt là do những kỹ thuật mô phỏng đã được dùng cho nhiều bài toán sắp thứ tự khác nhau, như bài toán lập lịch hay bài toán phân hoạch. Một số bài toán này sẽ được nói đến trong chương kế tiếp.

Có một điểm thống nhất trong cộng đồng GA là biểu diễn nhị phân của các hành trình không thích hợp với TSP. Lý do thật dễ hiểu: sau cùng thì ta muốn có một hoán vị tốt nhất của các thành phố, nghĩa là:

$(i_1, i_2, ..., i_n)$

trong đó, $(i_1, i_2, ..., i_n)$ là hoán vị của $\{1, 2, ..., n\}$. Mã nhị phân cho những thành phố này sẽ không thuận lợi. Ngược lại, biểu diễn nhị phân cần đến các thuật giải sửa chữa đặc biệt, do thay đổi của một bit duy nhất cũng có thể đưa đến một hành trình bất hợp lệ.

Trong một vài năm gần đây đã có 3 biểu diễn vectơ được xem xét có liên quan với TSP: biểu diễn *kề, bậc* và *lộ trình*. Mỗi biểu diễn này có các toán tử "di truyền" riêng – ta sẽ lần lượt bàn về chúng. Do tương đối dễ để tìm được một loại toán tử đột biến nào đó có thể đưa vào một thay đổi nhỏ trong hành trình, nên ta chủ yếu tập trung vào các toán tử lai. Trong cả ba biểu diễn này, hành trình được mô tả như một danh sách các thành phố. Trong những phần sau ta dùng một thí dụ có 9 thành phố được đánh số từ 1 đến 9.

BIỂU DIỄN KỀ

Biểu diễn kề biểu diễn một hành trình là danh sách n thành phố. Thành phố j được đặt trong danh sách ở vị trí i nếu và chỉ nếu chuyến đi bắt đầu từ thành phố i đến thành phố j. Thí dụ, vectơ:

(2 4 8 3 9 7 1 5 6)

biểu diễn hành trình sau đây:

1 – 2 – 4 – 3 – 8 – 5 – 9 – 6 – 7

Mỗi thành phố chỉ xuất hiện 1 lần trong danh sách kề; nhưng một số danh sách kề có thể biểu diễn các hành trình bất hợp lệ, nghĩa là:

(2 4 8 1 9 3 5 7 6),

dẫn đến:

1 – 2 – 4 – 1,

nghĩa là, một (phần) hành trình có một chu trình (xảy ra quá sớm).

Biểu diễn kề không hỗ trợ toán tử lai cổ điển. Có lẽ cần có một thuật giải sửa chữa. Ba toán tử lai được định nghĩa cho biểu diễn kề: *lai có các cạnh thay đổi, lai các đoạn hành trình con, và lai heuristic*.

- Lai–cạnh thay đổi, tạo ra con mới bằng cách chọn (ngẫu nhiên) một cạnh từ cha–mẹ thứ nhất, rồi chọn một cạnh thích hợp từ cha–mẹ thứ hai, vv... – toán tử mở rộng hành trình bằng cách chọn các cạnh từ các cha–mẹ luân phiên. Nếu cạnh mới từ một trong các cha–mẹ tạo một chu trình vào hành trình hiện hành (mới một phần), thì toán tử chọn một cạnh thay thế (ngẫu nhiên) từ các cạnh còn lại không tạo các chu trình. Thí dụ, con thứ nhất từ hai cha–mẹ:

  p_1 = (2 3 8 7 9 1 4 5 6) và

  p_2 = (7 5 1 6 9 2 8 4 3)

  có thể là:

  o_1 = (2 5 8 7 9 1 6 4 3), ở đây tiến trình khởi đầu từ cạnh (1, 2) của cha–me p_1, và cạnh ngẫu nhiên duy nhất được tạo trong tiến trình xen kẽ các cạnh là (7, 6) thay vì (7, 8) sẽ tạo ra một chu trình quá sớm.

- Lai–các đoạn hành trình con, tạo ra con mới bằng cách chọn hành trình con (chiều dài ngẫu nhiên) từ một trong các cha–mẹ, rồi chọn hành trình con (chiều dài ngẫu nhiên) từ một cha–mẹ khác, vv...– toán tử mở rộng hành trình bằng cách chọn các cạnh từ các cha–mẹ luân phiên. Lần nữa, nếu một cạnh nào đó (từ một trong các cha–mẹ tạo một chu trình vào hành trình hiện hành (mới một phần), thì toán tử chọn một cạnh thay thế (ngẫu nhiên) từ các cạnh còn lại không tạo các chu trình..
- Lai–heuristic, tạo ra con mới bằng cách chọn một thành phố ngẫu nhiên làm điểm khởi hành cho hành trình của con. Rồi so sánh hai cạnh (từ cả hai cha–mẹ) rời thành phố này và chọn cạnh (ngắn hơn) tốt hơn. Thành phố ở đầu bên kia của cạnh được chọn dùng làm điểm khởi hành trong việc chọn cạnh ngắn hơn trong hai cạnh rời thành phố này vv.... Nếu, ở một giai đoạn nào đó, cạnh mới đưa một chu trình vào hành trình, thì hành trình sẽ được mở rộng bởi một cạnh ngẫu nhiên từ các cạnh còn lại không tạo chu trình.

Tác động của toán tử này là để dán những đường dẫn con ngắn của các hành trình cha–mẹ. Nhưng, nó có thể để lại những giao điểm không muốn có của các cạnh – đó là lý do mà lai–heuristic không thích hợp cho việc dò tìm chính xác các hành trình. Suh và Gucht đưa vào một toán tử heuristic bổ sung (dựa trên thuật giải 2–opt) thích hợp cho việc dò tìm chính xác. Toán tử này chọn ngẫu nhiên hai cạnh $(i\ j)$ và $(k\ m)$, và kiểm tra xem:

$$dist(i, j) + dist(k, m) > dist(i, m) + dist(k, j)$$

trong đó $dist(a, b)$ là khoảng cách giữa hai thành phố a và b. Nếu đây là trường hợp các cạnh $(i\ j)$ và $(k\ m)$, trong hành trình được thay thế bởi các cạnh $(i\ m)$ và $(k\ j)$.

Một thuận lợi của biểu diễn kề là ta có thể sử dụng lý thuyết lược đồ (xem chương 3) để phân tích. Các lược đồ tương ứng với các khối kiến trúc tự nhiên, nghĩa là các cạnh; như lược đồ sau:

(* * * 3 * 7 * * *)

biểu thị tập của tất cả các hành trình với các cạnh (4 3) và (6 7). Nhưng bất lợi chính của biểu diễn này là các kết quả tương đối xấu đối với tất cả các toán tử. Lai–các cạnh xen kẽ thường làm rối loạn các hành trình tốt do thao tác của chính nó qua các cạnh xen kẽ từ hai cha–mẹ. Lai–đoạn hành trình con thực hiện tốt hơn lai–các cạnh xen kẽ, do tỉ lệ rối loạn thấp hơn. Nhưng, kết quả vẫn rất thấp. Dĩ nhiên lai–heuristic là toán tử tốt nhất ở đây là vì hai phép lai đầu tiên là mù quáng, nghĩa là chúng không để ý đến các chiều dài thực của các cạnh. Mặt khác, lai–heuristic chọn cạnh tốt hơn trong hai cạnh – vì vậy mà nó thực hiện tốt hơn hai cách lai kia. Nhưng, kết quả của lai–heuristic vẫn không phải là xuất sắc: trong nhiều thử nghiệm trên 50, 100, và 200 thành phố hệ thống tìm các hành trình cỡ 25%, 15%, và 27% tối ưu, trong lần lượt khoảng 15000, 20000, và 25000 thế hệ.

BIỂU DIỄN THỨ TỰ

Biểu diễn thứ tự biểu diễn hành trình là danh sách của n thành phố; phần tử thứ i của danh sách là một số trong khoảng từ 1 đến n–i+1. Ý nghĩa của biểu diễn thứ tự như sau. Có một danh sách thứ tự của các thành phố C, được dùng làm điểm tham chiếu cho các danh sách trong các biểu diễn thứ tự. Ví dụ:

C= (1 2 3 4 5 6 7 8 9)

Một hành trình:

1 – 2 – 4 – 3 – 8 – 5 – 9 – 6 – 7

được biểu diễn là danh sách *l* của các tham chiếu:

l = (1 1 2 1 4 1 3 1 1),

và phải được thông dịch như sau:

- Số thứ nhất trong danh sách *l* là 1, vì thế lấy thành phố thứ nhất trong danh sách C là thành phố thứ nhất của hành trình (thành phố số 1), và lấy ra khỏi *C*. Hành trình một phần là:

  1

- Số kế tiếp trong danh sách *l* cũng là 1, vì thế lấy thành phố thứ nhất trong danh sách (còn lại) hiện hành *C* là thành phố kế tiếp của hành trình (thành phố số 2), và lấy ra khỏi *C*. Hành trình một phần bây giờ là:

  1–2

- Số kế tiếp trong danh sách *l* là 2, vì thế lấy thành phố thứ hai trong danh sách hiện hành *C* là thành phố kế tiếp của hành trình (thành phố số 4), và lấy ra khỏi *C*. Hành trình một phần hiện hành là:

  1– 2 – 4

- Số kế tiếp trong danh sách *l* là 1, vì thế lấy thành phố thứ nhất trong danh sách hiện hành *C* là thành phố kế tiếp của hành trình (thành phố số 3), và lấy ra khỏi *C*. Hành trình một phần hiện hành là:

  1–2–4–3

- Số kế tiếp trong danh sách *l* là 4, vì thế lấy thành phố thứ tư trong danh sách hiện hành *C* là thành phố kế tiếp của hành trình (thành phố số 8), và lấy ra khỏi *C*. Hành trình một phần hiện hành là:

  1–2–4–3–8

- Số kế tiếp trong danh sách *l* lại là1, vì thế lấy thành phố thứ nhất trong danh sách hiện hành *C* là thành phố kế tiếp của hành trình (thành phố số 5), và lấy ra khỏi *C*. Hành trình một phần hiện hành là:

  1–2–4–3–8–5

- Số kế tiếp trong danh sách *l* là 3, vì thế lấy thành phố thứ ba trong danh sách hiện hành *C* là thành phố kế tiếp của hành trình (thành phố số 9), và lấy ra khỏi *C*. Hành trình một phần hiện hành là:

  1–2–4–3–8–5–9

- Số kế tiếp trong danh sách *l* là 1, vì thế lấy thành phố thứ hai trong danh sách hiện hành *C* là thành phố kế tiếp của hành trình (thành phố số 6), và lấy ra khỏi *C*. Hành trình một phần hiện hành là:

  1–2–4–3–8–5–9–6

- Số kế tiếp trong danh sách *l* là 2, vì thế lấy thành phố thứ hai trong danh sách hiện hành *C* là thành phố kế tiếp của hành trình (thành phố số 4), và lấy ra khỏi *C*. Hành trình một phần hiện hành là:

  1–2–4–3–8–5–9–6–7

Lợi ích chính của biểu diễn thứ tự là phép lai cổ điển hoạt động tốt! Hai hành trình bất kỳ trong biểu diễn thứ tự được cắt sau một vị

trí nào đó và giao nhau, sẽ tạo ra hai con, mỗi con là một hành trình hợp lệ. Thí dụ hai cha–mẹ:

p_1 = (1 1 2 1|4 1 3 1 1) và

p_2 = (5 1 5 5 |5 3 3 2 1),

tương ứng với các hành trình:

1–2–4–3–8–5–9–6–7 và

5–1–7–8–9–4–6–3–2,

với điểm lai tạo được đánh dấu bởi '|', tạo ra các con sau đây:

o_1 = (1 1 2 1 5 3 3 2 1) và

o_2= (5 1 5 5 4 1 3 1 1);

các con này tương ứng với:

1–2–4–3–9–7–8–6–5 và

5–1–7–8–6–2–9–3–4.

Rất dễ thấy là các hành trình một phần không thay đổi phía bên trái của điểm lai, trong khi các hành trình một phần bên phải điểm lai lại bị rối loạn theo cách hoàn toàn ngẫu nhiên. Các kết quả thử nghiệm cho thấy biểu diễn này cùng phép lai cổ điển không thích hợp cho TSP.

BIỂU DIỄN ĐƯỜNG DẪN

Có lẽ biểu diễn đường dẫn là biểu diễn tự nhiên nhất của một hành trình. Thí dụ hành trình

5–1–7–8–9–4–6–2–3

đơn giản được biểu diễn là:

(5 1 7 8 9 4 6 2 3)

cho đến gần đây, đã có ba phép lai được định nghĩa cho biểu diễn đường dẫn: lai–*được ánh xạ từng phần* (PMX), *thứ tự* (OX), và *chu trình* (CX). Ta lần lượt bàn về chúng.

- PMX – do Golberg và Lingle đề nghị – tạo ra con mới bằng cách chọn một chuỗi con của hành trình từ một cha–mẹ đồng thời bảo toàn thứ tự và vị trí của tối đa có thể các thành phố từ cha–mẹ kia. Một chuỗi con của hành trình được chọn bằng cách chọn hai điểm cắt ngẫu nhiên, được dùng làm hai giới hạn cho các thao tác hoán vị. Thí dụ, hai cha–mẹ (với hai điểm cắt được đánh dấu bởi '|')

  p_1 = (1 2 3 | 4 5 6 7 | 8 9) và

  p_2 = (4 5 2 | 1 8 7 6| 9 3)

  tạo ra con theo cách sau. Trước hết các đoạn giữa các điểm cắt được hoán vị ('x' được hiểu là 'hiện tại chưa biết'):

  o_1 = (x x x| 1 8 7 6| x x) và

  o_2 =(x x x| 4 5 6 7 | x x)

  Hoán vị này cũng định nghĩa một loạt các ánh xạ:

  1↔4, 8↔5, 7↔6, và 6↔7.

  Rồi ta có thể thêm vào các thành phố (từ các cha–mẹ gốc), mà không có xung đột:

  o_1 = (x 2 3| 1 8 7 6| x 9) và

o_2 =(x x 2| 4 5 6 7 | 9 3)

Cuối cùng, x đầu tiên trong con o_1 (lẽ ra phải là 1, nhưng không có xung đột) được thay bằng 4, vì ánh xạ (1↔4). Tương tự, x thứ hai trong con o_1 được thay bằng 5, còn x và x trong con o_2 là 1 và 8. Các con là:

o_1 = (4 2 3| 1 8 7 6| 5 9) và

o_2 =(1 8 2| 4 5 6 7 | 9 3) .

Lai PMX khai thác các điểm tương đồng quan trọng trong giá trị và xếp bậc đồng thời khi được sử dụng với một kế hoạch sinh sản thích hợp.

- OX – do Davis đề nghị – tạo ra các con bằng cách chọn một chuỗi con của hành trình từ một cha–mẹ và bảo tồn thứ tự tương đối của các thành phố của cha–mẹ kia. Thí dụ, hai cha–mẹ (với hai điểm cắt được đánh dấu bởi '|')

  p_1 = (1 2 3 | 4 5 6 7 | 8 9) và

  p_2 = (4 5 2 | 1 8 7 6| 9 3)

  tạo ra các con như sau. Đầu tiên, các đoạn giữa các điểm cắt được sao chép vào các con:

  o_1 = (x x x| 4 5 6 7 | x x) và

  o_2 =(x x x| 1 8 7 6 | x x).

  Kế đến, bắt đầu từ điểm cắt thứ hai của một cha–mẹ, các thành phố từ cha–mẹ kia được sao chép theo cùng thứ tự, bỏ đi các dấu hiệu đã có. Đến cuối chuỗi, ta tiếp tục từ vị trí đầu tiên của chuỗi. Thứ tự của các thành phố trong cha–mẹ thứ hai (từ điểm cắt thứ hai là:

  9 – 3 – 4 – 5 – 2 – 1 – 8 – 7 – 6;

  sau khi bỏ đi các thành phố 4, 5, 6 và 7 đã có trong con thứ nhất, ta có:

  9–3–2–1–8.

  Thứ tự này được đặt vào con thứ nhất (bắt đầu từ điểm cắt thứ hai):

  o_1 = (2 1 8| 4 5 6 7 | 9 3).

  Tương tự ta có con kia là:

  o_2 =(3 4 5| 1 8 7 6 | 9 2).

  Lai tạo OX khai thác thuộc tính về biểu diễn đường dẫn, mà thứ tự của các thành phố (chứ không phải là vị trí của chúng) rất quan trọng, nghĩa là, hai hành trình,:

  9–3–4–5–2–1–8–7–6

  4–5–2–1–8–7–6–9–3

  thực ra là giống nhau.

- CX – do Oliver đề nghị – xây dựng con theo cách mỗi thành phố (và vị trí của nó) xuất phát từ một trong các cha–mẹ. Ta giải thích cơ chế của lai chu trình bằng thí dụ sau đây. Hai cha–mẹ:

  p_1 = (1 2 3 4 5 6 7 8 9) và

  p_2 = (4 1 2 8 7 6 9 3 5)

  sẽ tạo ra con thứ nhất bằng cách lấy thành phố thứ nhất từ cha–mẹ thứ nhất:

  o_1= (1 x x x x x x x x)

  Do mỗi thành phố trong con có thể bị lấy đi từ một trong các cha–mẹ của nó (từ cùng vị trí), nên giờ ta không có chọn lựa nào: thành phố kế tiếp được xét phải là thành phố 4, do thành phố từ cha–mẹ p_2 ngay "bên

dưới" thành phố 1 được chọn. Trong p_1 thành phố này ở tại vị trí '4', như vậy:

o_1= (1 x x 4 x x x x x).

Điều này, trở lại bao hàm thành phố 8, do thành phố từ cha–mẹ p_2 ngay "bên dưới" thành phố 4 được chọn. Như vậy:

o_1= (1 x x 4 x x x 8 x);

Theo luật này, các thành phố kế tiếp được gộp vào con thứ nhất là 3 và 2. Nhưng chú ý rằng việc chọn thành phố 2 đòi hỏi việc chọn thành phố 1, đã có trong danh sách – như vậy ta đã hoàn thành một chu trình:

o_1= (1 2 3 4 x x x 8 x).

Các thành phố còn lại được hoàn thành từ cha–mẹ kia:

o_1= (1 2 3 4 7 6 9 8 5).

Tương tự:

o_2 = (4 1 2 8 5 6 7 3 9).

CX bảo toàn vị trí tuyệt đối của các phần tử theo thứ tự của cha–mẹ.

Có thể định nghĩa các toán tử khác cho biểu diễn đường dẫn. Chẳng hạn, Syswerda định nghĩa hai phiên bản hiệu chỉnh của toán tử lai thứ tự. (Nhưng công trình này liên quan với bài toán lập lịch sẽ bàn đến trong chương sau) Hiệu chỉnh thứ nhất (là lai dựa trên thứ tự) chọn (ngẫu nhiên) nhiều vị trí trong một vectơ, và thứ tự trong các thành phố trong các vị trí được chọn trong một cha–mẹ được đặt trên các thành phố tương ứng trong cha–mẹ kia. Thí dụ, xét hai cha–mẹ:

p_1 = (1 2 3 4 5 6 7 8 9) và

p_2 = (4 1 2 8 7 6 9 3 5).

Giả sử các vị trí được chọn là thứ 3, thứ 4, thứ 6 và thứ 9; việc xếp thứ tự các thành phố trong những vị trí này từ cha–mẹ p_2 sẽ được áp đặt trên cha–mẹ p_1. Các thành phố tại những vị trí này (theo thứ tự đã cho) trong p_2 là 2, 8, 6 và 5. Trong cha–mẹ p_1 các thành phố này xuất hiện tại các vị trí 2, 5, 6 và 8. Trong các con của các phần tử tại những vị trí này được sắp xếp lại để phù hợp với thứ tự của cùng những phần tử này từ p_2 (thứ tự là 2 – 8 – 6 – 5). Con đầu tiên là bản sao của p_1 trên tất cả vị trí trừ các vị trí 2, 5, 6, và 8:

o_1= (1 x 3 4 x x 7 x 9).

tất cả những phần tử khác được đưa vào đầy đủ theo thứ tự đã cho trong cha–mẹ p_2, nghĩa là 2, 8, 6, 5, và cuối cùng:

o_1 = (1 2 3 4 8 6 7 5 9).

Tương tự, ta có thể xây dựng con thứ hai:

o_2 = (3 1 2 8 7 4 6 9 5).

Hiệu chỉnh thứ hai (lai trên vị trí) giống với lai thứ tự gốc nhiều hơn. Khác biệt duy nhất là trong lai trên vị trí, thay vì chọn một chuỗi con các thành phố để sao chép, thì nhiều thành phố lại được chọn (ngẫu nhiên) cho mục đích này.

Khá thú vị khi nhận thấy rằng hai toán tử này (lai trên thứ tự và lai trên vị trí) theo một ý nghĩa nào đó là tương đương nhau. Lai trên thứ tự với một số vị trí được chọn làm các điểm lai, và lai trên vị trí với những vị trí đáng khen là các điểm lai của nó sẽ luôn cho ra cùng kết quả. Điều này có nghĩa là nếu số các điểm lai trung bình là $m/2$ (m là tổng số thành phố), thì hai toán tử này sẽ thực hiện

giống nhau. Nhưng, nếu số các điểm lai tạo trung bình là, $m/10$ chẳng hạn, thì hai toán tử này sẽ cho những kết quả khác nhau.

Trong việc nghiên cứu những toán tử tái sắp xếp xuất hiện trong một vài năm gần đây, ta cũng nên nói đến toán tử đảo. Phép đảo đơn giản chọn hai điểm theo chiều dài của nhiễm sắc thể, được cắt tại những điểm này, và chuỗi con giữa các điểm này bị đảo. Thí dụ, một nhiễm sắc thể:

(1 2 3 | 4 5 6 7 | 8 9)

với hai điểm cắt đánh dấu bởi '|', được đổi thành:

(1 2 | 6 5 4 3 | 7 8 9).

Đảo đơn giản như thế bảo đảm rằng đứa con có được là một hành trình hợp lệ; một số phát hiện về lý thuyết cho thấy rằng toán tử phải có ích trong việc tìm ra những cách sắp xếp chuỗi tốt. Có người rằng trong một TSP có 50 thành phố, một hệ thống có đảo thực hiện trội hơn hệ thống có toán tử "giao và sửa". Nhưng, việc tăng số điểm cắt làm giảm hiệu quả của hệ thống. Cũng vậy, đảo (giống như đột biến) là một toán tử một ngôi, chỉ có thể bổ sung cho những toán tử tái kết hợp, toán tử này không thể tự tái kết hợp thông tin. Nhiều phiên bản của toán tử đảo đã được khám phá. Holland cũng đã cung cấp một thay đổi về Định lý Lược đồ để thêm hiệu quả.

Ở điểm này ta cũng có thể nhắc đến một số nỗ lực gần đây, dùng các chiến lược tiến hóa để giải TSP. Một trong những cố gắng này đã được thử nghiệm với 4 toán tử đột biến khác nhau:

- *Đảo* – như mô tả ở trên;
- *Chèn* – chọn một thành phố và chèn nó vào một vị trí ngẫu nhiên;

- *Dời chỗ* – chọn một hành trình con rồi chèn nó vào một vị trí ngẫu nhiên;
- *Trao đổi qua lại*– hoán vị hai thành phố.

Một phiên bản của toán tử lai–heuristic cũng được sử dụng. Trong hiệu chỉnh này, nhiều cha–mẹ góp sức trong việc tạo ra con. Sau khi chọn thành phố đầu tiên trong hành trình của các con (ngẫu nhiên), tất cả lân cận bên phải và trái của thành phố đó (từ tất cả cha–mẹ) đều được khảo sát. Thành phố có khoảng cách ngắn nhất sẽ được chọn. Tiến trình tiếp tục cho đến khi hoàn tất hành trình.

Một ứng dụng khác của chiến lược tiến hóa là phát sinh một vectơ (số chấm động) trên n số (n tương ứng với số thành phố). Chiến lược tiến hóa được áp dụng như cho bất kỳ một bài toán liên tục nào. Kỹ xảo là ở việc mã hóa. Các thành phần của vectơ được sắp xếp và thứ tự của chúng quyết định hành trình. Thí dụ vectơ:

$v = (2.34, -1.09, 1.91, 0.87, -0.12, 0.99, 2.13, 1.23, 0.55)$

tương ứng với hành trình:

2 – 5 – 9 – 4 – 6 – 8 – 3 – 7 – 1,

do số nhỏ nhất: –1.09 là thành phần thứ hai của vectơ v, nên số nhỏ thứ hai: 0.12 là thành phần thứ năm của vectơ vv,...

Hầu hết các toán tử được nói đến từ trước đến nay đều quan tâm đến các thành phố (nghĩa là, các vị trí và thứ tự của chúng) ngược lại với các cạnh – các cung nối giữa các thành phố. Vị trí cụ thể của một thành phố trong hành trình không quan trọng, mà là các nối giữa thành phố này với các thành phố khác.

Grefenstette đã phát triển một lớp các toán tử heuristic chú trọng đến các cạnh. Các toán tử này làm việc như sau:

1. Chọn ngẫu nhiên thành phố của các con làm thành phố *c* hiện hành;
2. Chọn 4 cạnh (chọn 2 cạnh từ từng cha–mẹ) gắn liền với thành phố hiện hành *c*,
3. Xác định phân bố xác suất trên các cạnh được chọn, dựa trên chi phí của chúng. Xác suất của cạnh có liên quan đến thành phố được thăm trước đây là 0,
4. Chọn một cạnh. Nếu ít nhất một cạnh có xác suất khác 0, việc chọn được dựa vào phân bố ở trên; ngược lại, chọn ngẫu nhiên (từ những thành phố chưa thăm),
5. Thành phố ở 'đầu kia' của cạnh được chọn trở thành thành phố hiện hành *c*,
6. Nếu hành trình hoàn tất, dừng; ngược lại trở về bước 2.

Tuy vậy, những toán tử như thế chuyển khoảng 60% các cạnh từ các cha–mẹ – điều này có nghĩa là 40% các cạnh được chọn ngẫu nhiên.

Whitley, Starweather và Fuquay đã phát triển một toán tử lai tạo mới: lai tạo tái kết hợp cạnh (ER), chuyển hơn 95% các cạnh từ các cha–mẹ cho đứa con duy nhất. Toán tử ER khai thác thông tin trên các cạnh trong một hành trình, như đối với hành trình:

(3 1 2 8 7 4 6 9 5),

các cạnh là (3 1), (1 2), (2 8), (8 7), (7 4), (4 6), (6 9), (9 5) và (5 3). Cuối cùng, các cạnh – không phải các thành phố – mang các giá trị (các khoảng cách) trong TSP. Hàm mục tiêu cần cực tiểu hóa là tổng các cạnh xây dựng một hành trình hợp lệ. Vị trí của một thành phố trong hành trình không quan trọng: Các hành trình là khép kín. Hướng của một cạnh cũng không quan trọng: cả hai cạnh (3 1) và (1 3) đều chỉ cho thấy rằng các thành phố 1 và 3 được nối trực tiếp.

Ý tưởng chính của lai ER là một con duy nhất được xây dựng từ các cạnh hiện diện trong cả hai cha–mẹ. Điều này được thực hiện với sự trợ giúp của danh sách cạnh được tạo từ các hành trình của cả hai cha–mẹ. Danh sách cạnh cung cấp, cho mỗi thành phố *c*, tất cả các thành phố khác nối với thành phố *c* trong ít nhất là một cha–mẹ. Rõ ràng, có ít nhất là hai và nhiều nhất là 4 thành phố trong danh sách cho mỗi thành phố *c*. Thí dụ, đối với hai cha–mẹ,

p_1 = (1 2 3 4 5 6 7 8 9) và

p_2 = (4 1 2 8 7 6 9 3 5),

danh sách cạnh là:

thành phố 1: có các cạnh đến các thành phố khác là: 9 2 4

thành phố 2: có các cạnh đến các thành phố khác là: 1 3 8

thành phố 3: có các cạnh đến các thành phố khác là: 2 4 9 5

thành phố 4: có các cạnh đến các thành phố khác là: 3 5 1

thành phố 5: có các cạnh đến các thành phố khác là: 4 6 3

thành phố 6: có các cạnh đến các thành phố khác là: 5 7 9

thành phố 7: có các cạnh đến các thành phố khác là: 6 8

thành phố 8: có các cạnh đến các thành phố khác là: 7 9 2

thành phố 9: có các cạnh đến các thành phố khác là: 8 1 6 3.

Việc xây dựng con bắt đầu bằng việc chọn một thành phố khởi tạo từ một cha–mẹ. Các tác giả đã chọn một trong những thành phố khởi tạo (thí dụ 1 hoặc 4 trong thí dụ trên). Thành phố có số cạnh nhỏ nhất trong danh sách cạnh sẽ được chọn. Nếu những số này bằng nhau thì chọn ngẫu nhiên. Việc chọn như thế làm tăng cơ hội hoàn thành hành trình với tất cả các cạnh được chọn từ các cha–mẹ.Với việc chọn ngẫu nhiên, cơ hội để xảy ra thất bại về cạnh, nghĩa là, ở bên trái với một thành phố không có cạnh đi tiếp, sẽ cao

hơn nhiều. Giả sử ta đã chọn thành phố 1. Thành phố này được nối trực tiếp với 3 thành phố khác: 9, 2, và 4. Thành phố kế tiếp sẽ được chọn từ 3 thành phố này. Trong thí dụ của ta, các thành phố 4 và 2 có 3 cạnh còn thành phố 9 có 4 cạnh. Có một chọn lựa ngẫu nhiên được thực hiện giữa các thành phố 4 và 2; giả sử thành phố 4 được chọn. Lần nữa, các ứng viên của thành phố kế tiếp trong hành trình được xây dựng là 3 và 5, do chúng được nối trực tiếp với thành phố cuối cùng, 4. Thành phố 5 lại được chọn, do nó chỉ có 3 cạnh trái với thành phố 3 có 4 cạnh. Cho đến giờ, con có dạng sau đây:

(1 4 5 x x x x x x).

Tiếp tục thủ tục này, ta chấm dứt với con,

(1 4 5 6 7 8 2 3 9),

được tạo thành hoàn toàn từ các cạnh lấy từ hai cha–mẹ. Từ một loạt các thử nghiệm, thất bại cạnh xuất hiện với tỉ lệ rất thấp (1%–1.5%).

Toán tử ER được thử nghiệm trên 3 bài toán TSP với 30, 50, và75 thành phố – trong tất cả các trường hợp nó trả về một lời giải tốt hơn kết quả "tốt nhất được biết đến" trước đây.

Hai năm sau đó, lai tái kết hợp cạnh được mở rộng. Ý tưởng là các 'chuỗi con thông thường' không được bảo toàn trong lai ER. Thí dụ, nếu danh sách cạnh (ở trên) chứa hàng có 3 cạnh:

Thành phố 4: có các cạnh đến các thành phố khác là: 3 5 1,

một trong những cạnh này sẽ lặp lại. Nói về thí dụ ở trên, đó là cạnh (4 5). Cạnh này có trong cả hai cha–mẹ. Nhưng nó được liệt kê là các cạnh khác, như (4 3) và (4 1), chỉ có trong một cha–mẹ. Lời giải được đề nghị hiệu chỉnh danh sách cạnh bằng cách lưu trữ các thành phố 'có cờ',

Thành phố 4: có các cạnh đến các thành phố khác là: 3 – 5 1;

Ký tự '–' chỉ đơn giản có nghĩa là thành phố có cờ 5 phải được liệt kê hai lần. Trong thí dụ trước đây về hai cha–mẹ:

p_1 = (1 2 3 4 5 6 7 8 9) và

p_2 = (4 1 2 8 7 6 9 3 5) ,

danh sách cạnh (nâng cao) là

Thành phố 1: có các cạnh đến các thành phố khác là: 9 –2 4

Thành phố 2: có các cạnh đến các thành phố khác là: –1 3 8

Thành phố 3: có các cạnh đến các thành phố khác là: 2 4 9 5

Thành phố 4: có các cạnh đến các thành phố khác là: 3 –5 1

Thành phố 5: có các cạnh đến các thành phố khác là: –4 6 3

Thành phố 6: có các cạnh đến các thành phố khác là: 5 –7 9

Thành phố 7: có các cạnh đến các thành phố khác là: –6 –8

Thành phố 8: có các cạnh đến các thành phố khác là: –7 9 2

Thành phố 9: có các cạnh đến các thành phố khác là: 8 1 6 3.

Thuật giải tạo con mới ưu tiên cho những thành phố có cờ: điều này chỉ quan trọng trong những trường hợp mà 3 cạnh được liệt kê – trong hai trường hợp kia, hoặc không thành phố nào có cờ, hoặc cả hai đều có. Mở rộng này (cộng với hiệu chỉnh để tạo những chọn lựa tốt hơn khi cần chọn cạnh ngẫu nhiên) cải thiện đáng kể hiệu quả của hệ thống.

Các toán tử tái kết hợp cạnh cho thấy rõ rằng biểu diễn đường dẫn có thể quá xấu để biểu diễn các thuộc tính quan trọng của một hành trình – đây là lý do mà nó được bổ sung bởi danh sách cạnh. Có những biểu diễn khác, thích hợp hơn cho bài toán người du lịch không ? Ta không thể khẳng định 'có' cho câu hỏi này. Nhưng cũng

đáng để thử nghiệm với những toán tử khác, với biểu diễn không phải vectơ.

Trong hai năm gần đây, có ít nhất 3 chương trình tiến hóa dựa trên biểu diễn ma trận của các nhiễm sắc thể cho TSP. Đó là những nỗ lực của Fox và McMahon, Seniw, và Homaifar và Guan. Ta sẽ lần lượt bàn ngắn gọn về chúng.

Fox và McMahon biểu diễn một hành trình là một ma trận nhị phân thứ tự ưu tiên $M = [m_{ij}]$. Phần tử m_{ij} trong hàng i và cột j chứa một số 1 nếu và chỉ nếu thành phố i xuất hiện trước thành phố j trong hành trình. Thí dụ, hành trình:

(3 1 2 8 7 4 6 9 5)

được biểu diễn theo dạng ma trận trong hình 8.1

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	0	1	0	1	1	1	1	1	1
2	0	0	0	1	1	1	1	1	1
3	1	1	0	1	1	1	1	1	1
4	0	0	0	0	1	1	0	0	1
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	1
7	0	0	0	1	1	1	0	0	1
8	0	0	0	1	1	1	1	0	1
9	0	0	0	0	1	0	0	0	0

Hình 8.1. Biểu diễn ma trận của hành trình

Trong biểu diễn này, ma trận $M_{n \times n}$ biểu diễn một hành trình có các thuộc tính sau đây:

(1) $\frac{n(n-1)}{2}$ là số các số 1 trong m

(2) $m_{ii} = 0$ với mọi $1 \le i \le n$, và

(3) nếu $m_{ij} = 1$ và $m_{jk} = 1$ thì $m_{ik} = 1$.

Nếu các số 1 trong ma trận ít hơn $\frac{n(n-1)}{2}$, và hai yêu cầu kia được thỏa thì các thành phố được sắp xếp một phần. Điều này có nghĩa là ta có thể hoàn tất ma trận đó (ít nhất theo một cách) để có được hành trình hợp lệ.

Toán tử giao dựa trên các quan sát rằng phép giao các bit từ cả hai ma trận kết quả trong ma trận mà (1) số số 1 không lớn hơn $\frac{n(n-1)}{2}$ và (2) cả hai cần được thỏa. Thế nhưng, ta có thể bổ túc một ma trận như thế để có được một hành trình hợp lệ.

Thí dụ, hai cha–mẹ:

p_1 = (1 2 3 4 5 6 7 8 9) và p_2 = (4 1 2 8 7 6 9 3 5)

được biểu diễn bằng hai ma trận (hình 8.2).

Phép giao hai ma trận này cho ra ma trận trình bày trong hình 8.3.

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	1	1	1	1	1
2	0	0	1	1	1	1	1	1	1
3	0	0	0	1	1	1	1	1	1
4	0	0	0	0	1	1	1	1	1
5	0	0	0	0	0	1	1	1	1
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	0	1	1
8	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0

Hình 8.2.a. Hai cha–mẹ

	1	2	3	4	5	6	7	8	9
1	0	1	1	0	1	1	1	1	1
2	0	0	1	0	1	1	1	1	1
3	0	0	0	0	1	0	0	0	0
4	1	1	1	0	1	1	1	1	1
5	0	0	0	0	0	0	0	0	0
6	0	0	1	0	1	0	0	0	1
7	0	0	1	0	1	1	0	0	1
8	0	0	1	0	1	1	1	0	1
9	0	0	1	0	1	0	0	0	0

Hình 8.2.b. Hai cha–mẹ

	1	2	3	4	5	6	7	8	9
1	0	1	1	0	1	1	1	1	1
2	0	0	1	0	1	1	1	1	1
3	0	0	0	0	1	0	0	0	0
4	0	0	0	0	1	1	1	1	1
5	0	0	0	0	0	1	1	1	1
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	0	1	1
8	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0

Hình 8.3. Giai đoạn thứ nhất của toán tử giao.

Thứ tự một phần được áp đặt bởi kết quả của phép giao đòi hỏi thành phố 1 đứng trước các thành phố 2, 3, 5, 6, 7, 8, và 9; thành phố 2 đứng trước các thành phố 3, 5, 6, 7, 8, và 9; thành phố 3 đứng trước thành phố 5; thành phố 4 đứng trước các thành phố 5, 6, 7, 8, và 9; còn các thành phố 6, 7, và 8 đứng trước thành phố 9.

Trong giai đoạn kế tiếp của toán tử giao, một trong số các cha-mẹ được chọn; vài số 1 (độc nhất đối với cha–mẹ này) được 'thêm vào', và ma trận được hoàn tất theo một trình tự qua một phân tích của tổng số các hàng và các cột. Thí dụ, ma trận trong hình 8.4 là



một kết quả sau khi hoàn tất giai đoạn 2; nó biểu diễn hành trình (1 2 4 8 7 6 3 5 9).

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	1	1	1	1	1
2	0	0	1	1	1	1	1	1	1
3	0	0	0	0	1	0	0	0	1
4	0	0	1	0	1	1	1	1	1
5	0	0	0	0	0	0	0	0	1
6	0	0	1	0	1	0	0	0	1
7	0	0	1	0	1	1	0	0	1
8	0	0	1	0	1	1	1	0	1
9	0	0	0	0	0	0	0	0	0

Hình 8.4. Kết quả chung cuộc của phép giao

Toán tử độc nhất được dựa trên quan sát rằng tập con các bit từ một ma trận có thể được kết hợp an toàn với tập con các bit từ ma trận kia, nghĩa là hai tập con này có phần giao là trống. Toán tử chia tập các thành phố thành hai nhóm tách biệt. Đối với nhóm thành phố thứ nhất, nó sao các bit từ ma trận thứ nhất và đối với nhóm thành phố thứ hai nó sao chép các bit từ ma trận thứ hai. Cuối cùng, nó hoàn tất ma trận vào một trình tự qua một phân tích của tổng số các hàng và các cột (cũng như đối với toán tử giao).



Thí dụ, hai cha–mẹ p_1 và p_2 và việc chia các thành phố thành {1, 2, 3, 4} và {5, 6, 7, 8, 9} tạo ra ma trận như trong hình 8.5, được hoàn tất cũng như đối với toán tử giao.

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	x	x	x	x	x
2	0	0	1	1	x	x	x	x	x
3	0	0	0	1	x	x	x	x	x
4	0	0	0	0	x	x	x	x	x
5	x	x	x	x	0	0	0	0	0
6	x	x	x	x	1	0	0	0	1
7	x	x	x	x	1	1	0	0	1
8	x	x	x	x	1	1	1	0	1
9	x	x	x	x	1	0	0	0	0

Hình 8.5. Thời kỳ đầu của toán tử hội

Các kết quả thực nghiệm trên nhiều cấu hình thành phố khác nhau (ngẫu nhiên, các nhóm, các vòng đồng tâm) bộc lộ một đặc trưng thú vị của các toán tử giao và hội, đặc trưng này tạo ra các tiến bộ ngay cả khi không sử dụng toán tử ưu tú (bảo toàn cái tốt nhất). Đây không phải là trường hợp của các toán tử ER hay PMX.

Phương pháp thứ hai sử dụng biểu diễn ma trận được Davis Seniw đề nghị. Phần tử ma trận m_{ij} trong hàng i và cột j chứa một

số 1 nếu và chỉ nếu hành trình đi trực tiếp từ thành phố *i* đến thành phố *j*. Điều này có nghĩa là chỉ có một ô khác 0 cho mỗi hàng và mỗi cột trong ma trận (đối với mỗi thành phố *i* có đúng một thành phố được thăm trước *i*, và đúng một thành phố được thăm kế *i*). Thí dụ, một nhiễm sắc thể trong hình 8.6 (a) biểu diễn một hành trình thăm các thành phố (1, 2, 4, 3, 8, 6, 5, 7, 9) theo thứ tự này. Cũng lưu ý rằng, biểu diễn này tránh bài toán đặc tả thành phố khởi đầu, nghĩa là, hình 8.6(a) cũng diễn tả các hành trình (2, 4, 3, 8, 6, 5, 7, 9, 1) và (4, 3, 8, 6, 5, 7, 9, 1, 2) vv...

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	0	1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	1	0	0	0
9	1	0	0	0	0	0	0	0	0

Hình 8.6.a. Các nhiễm sắc thể ma trận nhị phân

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	0	1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0
4	0	0	0	0	1	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	0	0	1
7	1	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	0
9	0	0	1	0	0	0	0	0	0

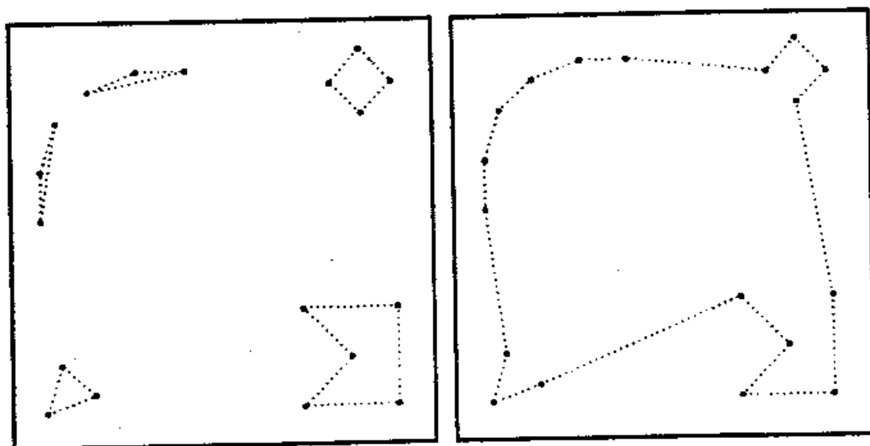
Hình 8.6.b.

Một nhận xét thú vị là mỗi hành trình trọn vẹn được biểu diễn thành một ma trận nhị phân chỉ có một bit trong mỗi hàng và một bit trong mỗi cột được thiết lập là 1. Tuy vậy, không phải tất cả các ma trận có những thuộc tính này đều có thể biểu diễn chỉ một hành trình. Các nhiễm sắc thể ma trận nhị phân có thể biểu diễn nhiều hành trình con. Mỗi hành trình con cuối cùng sẽ vòng trở lại chính nó, mà không nối với một hành trình con nào khác trong nhiễm sắc thể. Thí dụ, nhiễm sắc thể ở hình 8.6(b) biểu diễn hai hành trình con (1, 2, 4, 5, 7) và (3, 8, 6, 9).

Ta có thể sử dụng các hành trình con này với hy vọng việc xếp nhóm tự nhiên có thể xảy ra. Sau khi chương trình tiến hóa kết

thúc, nhiễm sắc thể tốt nhất được giảm thành một hành trình đơn bằng cách sử dụng một thuật giải tất định để kết hợp liên tiếp các cặp hành trình con. Các hành trình con của một thành phố (một chuyến đi rời thành phố để đi trở lại chính nó), có chi phí khoảng cách là 0 thì không được chấp nhận. Giới hạn thấp hơn của $q = 3$ thành phố trong một hành trình con được thiết lập hầu ngăn GA giảm bài toán TSP thành một số lớn các hành trình con, mà mỗi hành trình con chỉ có rất ít thành phố (q là một tham số của phương pháp).

Hình 8.7(a) miêu tả các hành trình con là kết quả của một lần chạy thử của thuật giải trên một số thành phố được cố ý đặt vào các nhóm. Như mong đợi thuật giải phát triển các hành trình con biệt lập. Hình 8.7 (b) vẽ ra hành trình sau khi các hành trình con đã được nối kết.



Hình 8.7. Các hành trình con riêng biệt và hành trình chung cuộc

Hai toán tử di truyền đã được định nghĩa: đột biến và lai. Toán tử đột biến lấy một nhiễm sắc thể, chọn ngẫu nhiên nhiều hàng và nhiều cột trong nhiễm sắc thể đó, bỏ đi các bit đã được thiếp lập

trong các phần giao của những hàng và cột đó, và thay thế chúng vào cấu hình khác, một cách ngẫu nhiên

Thí dụ, xét hành trình trong hình 8.6(a) biểu diễn hành trình sau:

(1, 2, 4, 3, 8, 6, 5, 7, 9).

Giả sử các hàng 4, 6, 7, 9 và các cột 1, 3, 5, 8, và 9 được chọn ngẫu nhiên để tham gia đột biến. Các tổng của các hàng và các cột ở biên được tính. Các bit ở các phần giao của các hàng và cột này được bỏ đi và thay thế một cách ngẫu nhiên, mặc dù chúng phải khớp với các tổng ở biên. Nói cách khác, ma trận con tương ứng với các hàng 4, 6, 7 và 9, và các cột 1, 3, 5, 8, và 9 từ ma trận gốc (hình 8.8 (a)), được thay bằng một ma trận con khác (hình 8.8(b)).

Nhiễm sắc thể kết quả biểu diễn nhiễm sắc thể với hai hành trình con:

(1, 2, 4, 5, 7) và (3, 8, 6, 9)

và được biểu diễn trong hình 8.6(b).

0	1	0	0	0
0	0	1	0	0
0	0	0	0	1
1	0	0	0	0

(a)

0	0	1	0	0
0	0	0	0	1
1	0	0	0	0
0	1	0	0	0

(b)

Hình 8.8. Phần nhiễm sắc thể trước (a) và sau (a) đột biến.

Toán tử lai bắt đầu bằng nhiễm sắc thể con có tất cả các bit được gán lại bằng 0. Toán tử thứ nhất khảo sát hai nhiễm sắc thể cha–mẹ, và khi nó khám phá cũng bit đó (cùng hàng và cột) được thiết lập (nghĩa là, 1) trong cả cha–mẹ, thì nó thiết lập bit tương ứng trong nhiễm sắc thể con (thời kỳ 1). Rồi toán tử luân phiên chép một bit được thiết lập từ mỗi cha–mẹ, cho đến khi không còn bit nào trong cả hai cha–mẹ có thể được chép mà không xâm phạm các giới hạn cơ bản của cấu trúc nhiễm sắc thể (thời kỳ 2). Cuối cùng, nếu không còn hàng nào trong nhiễm sắc thể của đứa con còn chứa một bit được thiết lập, nhiễm sắc thể sẽ được phủ đầy một cách ngẫu nhiên (thời kỳ cuối). Do phép lai theo truyền thống, tạo ra hai nhiễm sắc thể con, toán tử được thực thi lần thứ hai với các nhiễm sắc thể cha–mẹ được đổi chỗ.

Thí dụ về phép lai sau đây bắt đầu bằng nhiễm sắc thể cha–mẹ thứ nhất trong hình 8.9(a), biểu diễn hai hành trình con:

(1, 5, 3, 7, 8) và (2, 4, 9, 6).

và nhiễm sắc thể cha–mẹ thứ hai (Hình 8.9(b) biểu diễn một hành trình:

(1, 5, 6, 2, 7, 8, 3, 4, 9).

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	0	0	0	0	1	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	1	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	1	0	0	0

Hình 8.9.a. Cha–mẹ thứ nhất (a) và thứ hai (b)

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	1	0	0
3	0	0	0	1	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	0	0	1	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0

Hình 8.9.b.

Hai thời kỳ đầu tiên trong việc xây dựng con thứ nhất được trình bày trong hình 8.10.

Con thứ nhất để lai, sau thời kỳ cuối, được trình bày trong hình 8.11. Nó biểu diễn hành trình con:

(1, 5, 6, 2, 3,4, 9, 7, 8)

Con thứ hai diễn tả:

(1, 5, 3, 4, 9) và (2, 7, 8, 6).

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0

Hình 8.10.a. Các cha–mẹ thứ nhất

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0

Hình 8.10.*b*.

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	1	0	0

Hình 8.11. Các cha–mẹ thứ hai

Chú ý rằng có những đoạn chung của các nhiễm sắc thể cha–mẹ trong cả hai con.

Chương trình tiến hóa này cho kết quả hợp lý trong nhiều trường hợp thử nghiệm từ 300 đến 500 thành phố. Nhưng, không thấy rõ ràng ảnh hưởng của tham số q (số thành phố tối thiểu trong

một hành trình con) trên chất lượng của lời giải cuối cùng. Cũng vậy, các thuật giải kết hợp nhiều hành trình con vào một hành trình duy nhất cũng chẳng có gì rõ ràng. Mặt khác, phương pháp có một số điểm tương đồng với thuật giải xếp nhóm đệ quy của Litke, thuật giải này thay thế một cách đệ quy các nhóm có kích thước B bằng các thành phố đại diện cho đến khi còn ít hơn B thành phố. Rồi thì, bài toán nhỏ hơn được giải một cách tối ưu. Tất cả các nhóm lần lượt được mở rộng và thuật giải xếp thứ tự cho tập mở rộng giữa hai láng giềng trong hành trình hiện hành. Phương pháp này cũng có thể có ích cho việc giải bài toán nhiều người lái buôn du lịch, mà nhiều người lái buôn có thể hoàn thành các hành trình (không phủ nhau) riêng biệt của họ.

Phương pháp thứ ba dựa trên biểu diễn ma trận đã được Homaifar và Guan đề nghị. Cũng như phương pháp trước, phần tử m_{ij} của ma trận nhị phân *M* được thiết lập bằng 1, nếu và chỉ nếu có một cạnh từ thành phố *i* đến thành phố j. Nhưng, họ dùng các toán tử lai tạo khác và đảo heuristic.

Hai toán tử lai ma trận (MX) được định nghĩa như sau. Những toán tử này trao đổi tất cả các mục nhập của hai ma trận cha–mẹ hoặc sau một điểm lai tạo đơn (lai tạo 1– điểm) hay giữa hai điểm lai tạo (lai tạo 2–điểm). Một "thuật giải sửa chữa" bổ sung được chạy để (1) loại các chỗ trùng nhau, nghĩa là, để bảo đảm rằng mỗi hàng và mỗi cột chỉ có chính xác một 1, và (2) cắt và nối các chu trình (nếu có) nhằm sản xuất một hành trình hợp lệ.

Lai 2–điểm được minh họa qua thí dụ sau. Hai ma trận cha–mẹ cho trong hình 8.12; biểu diễn hai hành trình hợp lệ:

(1 2 4 3 8 6 5 7 9) và (1 4 3 6 5 7 2 8 9)

	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	1	0	0	0
9	1	0	0	0	0	0	0	0	0

Hình 8.12.a. Những nhiễm sắc thể ma trận nhị phân với các điểm lai tạo được đánh dấu

	1	2	3	4	5	6	7	8	9
1	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	1	0	0	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1
9	1	0	0	0	0	0	0	0	0

Hình 8.12.b.

Hai điểm lai được chọn; đây là các điểm giữa các cột 2 và 3 (điểm đầu tiên, và giữa các cột 6 và 7 (điểm thứ hai). Các điểm lai cắt ma trận theo chiều dọc: đối với mỗi ma trận, hai hàng đầu tiên thành lập phần đầu của sự cắt; các cột 3, 4, 5, 6 thuộc phần giữa; ba cột cuối thuộc phần thứ ba. Sau bước thứ nhất của toán tử MX 2–điểm, các mục nhập của cả hai ma trận được hoán đổi giữa các điểm lai (nghĩa là các mục nhập trong các cột 3, 4, 5 và 6). Kết quả trung gian được trình bày trong hình 8.13.

Cả hai con, (a) và (b) đều bất hợp lệ; nhưng tổng số các số 1 trong mỗi ma trận trung gian đều đúng (là 9). Bước đầu tiên của "thuật giải sửa chữa" dời đi vài số 1 trong các ma trận để mỗi hàng và mỗi cột chỉ có đúng một 1. Thí dụ, trong đứa con ở hình 8.13 (a) các số 1 trùng lặp xuất hiện trong các hàng 1 và 3. Thuật giải có thể chuyển ô $m_{14} = 1$ vào m_{84}, còn ô $m_{38} = 1$ vào m_{28}. Tương tự, trong con khác (hình 8.13 (b)) các số 1 trùng lặp xuất hiện trong các hàng 2 và 8. Thuật giải có thể chuyển ô $m_{24} = 1$ vào m_{34}, còn ô $m_{86} = 1$ vào m_{16}. Sau khi hoàn tất bước đầu tiên của thuật giải sửa chữa, con thứ nhất biểu diễn một hành trình (hợp lệ):

(1 2 8 4 3 6 5 7 9)

và con thứ hai biểu diễn hành trình gồm có hai hành trình con:

(1 6 5 7 2 8 9) và (3 4).

	1	2	3	4	5	6	7	8	9
1	0	1	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	0	1	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0

Hình 8.13.a. Hai con tức thời sau bước đầu tiên của toán tử MX

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	1	0	0	0	0	0	0	1
8	0	0	0	0	0	1	0	0	1
9	1	0	0	0	0	0	0	0	0

Hình 8.13.b.

Bước thứ hai của thuật giải sửa chữa chỉ nên áp dụng cho con thứ hai. Trong thời kỳ này, thuật giải cắt và nối các hành trình con để cho ra một hành trình hợp lệ. Giai đoạn cắt và nối xét đến các cạnh đang có trong các cha–mẹ gốc. Thí dụ, cạnh (2 4) được chọn đến nối hai hành trình con này, vì cạnh này hiện diện trong một cha–mẹ. Như vậy, hành trình trọn vẹn (con thứ hai hợp lệ) là:

(1 6 5 7 2 4 3 8 9).

Toán tử thứ hai được Homaifar và Guan sử dụng bổ sung cho phép lai MX là đảo heuristic. Toán tử đảo thứ tự các thành phố giữa hai điểm cắt (như đảo đơn giản đã thực hiện – ta đã bàn về điều này trong phần trước của chương 8). Nếu khoảng cách giữa hai điểm cắt lớn (đảo thứ tự cao), toán tử khai thác các nối kết giữa các đường dẫn 'tốt', ngược lại (đảo thứ tự thấp), toán tử sẽ thực hiện tìm kiếm cục bộ. Tuy vậy, có hai điểm khác biệt giữa các toán tử cổ điển và các toán tử đảo được đề nghị. Điều khác biệt thứ nhất là đứa con kết quả chỉ được chấp nhận nếu hành trình mới tốt hơn hành trình gốc. Khác biệt thứ hai là thủ tục đảo chọn chỉ một thành phố trong một hành trình và kiểm soát sự cải thiện cho những lần đảo của thứ tự 2 (thấp nhất có thể). Đảo thứ nhất đưa đến cải thiện được chấp nhận và thủ tục đảo chấm dứt. Ngược lại, sẽ xét đến trong thứ tự 3, và cứ thế.

Các kết quả được báo cáo thực nghiệm cho thấy rằng chương trình tiến hóa với các toán tử MX 2–điểm và các toán tử đảo thực hiện thành công trên những bài toán TSP có từ 30 – 100 thành phố. Trong thử nghiệm gần đây nhất, kết quả của thuật giải này đối với bài toán 320 thành phố chỉ khác với lời giải tối ưu 0.6%.

Trong chương này ta không cung cấp các kết quả chính xác của nhiều thực nghiệm đối với các cấu trúc dữ liệu và các toán tử 'di truyền' khác nhau, mà đã thực hiện một tổng quan về nhiều nỗ lực trong việc xây dựng chương trình tiến hóa thành công cho bài toán

TSP. Một trong những lý do là hầu hết các kết quả thực hiện được nói đến tùy thuộc nặng nề vào nhiều chi tiết (kích thước quần thể, số thế hệ, kích thước bài toán, vv...). Thêm nữa, liên hệ đến kích thước tương đối nhỏ của TSP (trên 100 thành phố).

Tuy nhiên, phần lớn chương này so sánh phương pháp được đề nghị với những phương pháp khác. Hai họ các trường hợp thử nghiệm đã được dùng cho những so sánh này:

- Chọn lựa ngẫu nhiên các thành phố. Ở đây, một công thức thực nghiệm đối với chiều dài mong đợi L của một hành trình TSP tối thiểu là có ích:

  $$L^* = k\sqrt{n.R}$$

  trong đó, n là số thành phố , R là diện tích của hộp vuông mà các thành phố được đặt ngẫu nhiên trong đó, còn k là hằng số thực nghiệm xấp xỉ 0.765.

- Sưu tập về các thành phố đã được xuất bản (một phần có những lời giải tối ưu), ftp.softlib.rice.edu, directory/pub/tsplib. Có hai tập tin tsplib.sh và tsplib.tar, lần lượt có độ lớn là 6Mb và 2 Mb.

Để có một bức tranh hoàn chỉnh về ứng dụng các kỹ thuật thuật giải di truyền vào TSP, ta nên báo cáo về công trình của các nhà nghiên cứu khác, đã sử dụng các GA để tối ưu hóa cục bộ của bài toán TSP. Các thuật giải tối ưu hóa cục bộ (2–opt, 3–opt, LinKernighan) cũng khá hiệu quả.

Các thuật giải tối ưu hóa cục bộ, với một hành trình cho trước (hiện hành), đặc tả một tập các hành trình lân cận và thay hành trình hiện hành bằng lân cận tốt nhất (có thể). Bước này được áp dụng cho đến khi chạm được tối ưu cục bộ. Thí dụ, thuật giải 2–opt định nghĩa các hành trình lân cận

mà ta có thể đạt được từ hành trình khác bằng cách chỉ hiệu chỉnh hai cạnh.

Các thuật giải tìm kiếm cục bộ được dùng làm cơ sở cho việc phát triển thuật giải tìm kiếm cục bộ di truyền, nó:

- Sử dụng một thuật giải tìm kiếm cục bộ để thay mỗi hành trình trong quần thể hiện hành (có kích thước μ) bằng một hành trình (tối ưu cục bộ),
- Mở rộng quần thể bằng λ hành trình bổ sung – các con của toán tử tái kết hợp được áp dụng vào một số hành trình trong quần thể hiện hành,
- Lần nữa, sử dụng một thuật giải tìm kiếm cục bộ để thay mỗi con trong số λ con trong quần thể mở rộng bằng một hành trình (tối ưu cục bộ),
- Giảm kích thước của quần thể mở rộng về kích thước gốc, μ, theo một số chọn lọc (sinh tồn của phần tử thích nghi nhất),
- Lặp lại ba bước cuối cho đến khi gặp một điều kiện dừng nào đó (quá trình tiến hóa).

Chú ý là có một số tương đồng giữa thuật giải tìm kiếm cục bộ di truyền và chiến lược tiến hóa ($\mu + \lambda$) (phụ lục 2). Như trong ($\mu + \lambda$)–ES, các cá thể μ tạo ra λ con và quần thể mới (mở rộng) của ($\mu + \lambda$) cá thể lại được giảm xuống còn μ cá thể bằng một tiến trình chọn lọc.

Thuật giải tìm kiếm cục bộ di truyền nói trên tương tự với thuật giải di truyền dành cho bài toán TSP được Muhlenbein, Gorges–Schleuter, và Kramer đề nghị trước đó, thuật giải này khuyến khích "tiến hóa thông minh" của các cá thể. Thuật giải này:

- Sử dụng một thuật giải tìm kiếm cục bộ (opt–2) để thay mỗi hành trình trong quần thể hiện hành bằng một hành trình (tối ưu cục bộ),

- Chọn phối ngẫu để ghép đôi (các cá thể trên trung bình có nhiều con hơn),
- Sinh sản (lai và đột biến),
- Tìm cực tiểu bằng từng cá thể (giảm, bộ giải toán, mở rộng),
- Lặp lại ba bước cuối cho đến khi gặp một điều kiện dừng nào đó (quá trình tiến hóa).

Lai được dùng trong thuật giải là phiên bản của lai thứ tự (OX). Ở đây hai cha–mẹ với hai điểm cắt (được đánh dấu bằng '|'):

p_1 = (1 2 3 | 4 5 6 7 | 8 9) và

p_2 = (4 5 2 | 1 8 7 6 | 9 3),

sẽ sinh con theo cách sau. Trước tiên, các đoạn giữa hai điểm cắt được sao chép vào các con:

o_1 = (x x x | 4 5 6 7 | x x) và

o_2 = (x x x | 1 8 7 6 | x x).

Kế tiếp, (thay vì bắt đầu từ điểm cắt thứ hai của một cha–mẹ như trong trường hợp của OX), các thành phố từ cha–mẹ kia được sao chép theo cùng thứ tự từ bắt đầu chuỗi, bỏ qua các dấu hiệu đã có:

o_1 = (2 1 8 | 4 5 6 7 | 9 3) và

o_2 = (2 3 4 | 1 8 7 6 | 5 9).

Các kết quả thực nghiệm ít ra cũng đáng khích lệ. Thuật giải tìm được một hành trình cho bài toán 530 thành phố và chiều dài

tìm được là 27702, trong vòng 0.06% của lời giải tối ưu (do Padberg và Rinaldi tìm được: 27686).

Gần đây có thêm hai phương pháp bổ sung. Phương pháp thứ nhất, do Craighurst và Martin tập trung vào việc khảo sát mối liên lạc giữa việc ngăn ngừa việc hỗn giao (xem phụ lục 1) và việc thực hiện của thuật giải di truyền đối với bài toán TSP. Các tác giả sử dụng một GA có các tính năng sau đây cho các thử nghiệm của họ: kích thước quần thể là 128, chọn lựa dựa trên thế hệ được đặt trên cơ sở xếp bậc, mà 128 con tranh đấu với 128 cha–mẹ, việc chọn lọc cho lai tạo MPX cũng dựa trên xếp bậc, leo đồi cục bộ, 2–opt) được thực hiện trên đứa con lúc tạo ra, tỉ lệ đột biến là 0.005, và điều kiện dừng là 500 thế hệ liên tiếp mà không có cải thiện. Phương pháp cho hỗn giao có tính hướng – gia đình: các tác giả chỉ quan tâm đến các tiền bối của hai cá thể được chọn để lai, và chúng đưa ra nhiều luật hỗn giao. Luật hỗn giao thứ *k* cấm ghép đôi của một cá thể với k – 1 tiền bối (nghĩa là, với k = 0) thì không có các giới hạn, với k =1 thì cá thể không thể ghép đôi với chính nó, còn với k = 2, nó không thể ghép đôi với chính nó, với cha–mẹ nó, với con cái, cũng không thể với anh chị em ruột của nó, và cứ thế). Nhiều thử nghiệm đã được thực hiện (trên 6 bài toán thử nghiệm (trong thư viện softlib.rice.edu đã trình bày ở phần đầu chương này) với số thành phố trong khoảng từ 48 – 101). Các kết quả cho thấy sự phụ thuộc qua lại mạnh và thú vị giữa các luật cấm hỗn giao và tỉ lệ đột biến: đối với những tỉ lệ đột biến thấp thì việc ngăn hỗn giao cải thiện các kết quả, nhưng nếu tỉ lệ đột biến cao, thì tầm quan trọng của cơ chế ngăn ngừa hỗn giao sẽ giảm cho đến khi (với những tỉ lệ đột biến cao ≈ 0.1) thì nó làm kết quả của hệ thống yếu đi. Cũng vậy, những luật khác nhau về ngăn ngừa hỗn giao không tác động

đến sự đa dạng của quần thể (mà sự tương đồng giữa hai cá thể được đo làm tỉ số đo sự khác biệt giữa tổng số cạnh trong một lời giải và số các cạnh chung giữa các lời giải, và tổng số cạnh trong một lời giải) theo cách đầy ý nghĩa. Kết luận cuối cùng là câu trả lời phủ định cho câu hỏi sau đây:" nhiều luật cấm hơn có tốt hơn không ?". Valenzuela và Jones đề nghị một phương pháp thú vị áp dụng thuật giải tiến hóa cho các bài toán tổ hợp khó; phương pháp của họ dựa trên ý tưởng về kỹ thuật chia để trị của các thuật giải di truyền Karp–Steele dành cho các bài toán TSP. Thuật giải di truyền tiến hóa chia để trị (Evolutionary Divide and Conquer – EDAC) của họ có thể được áp dụng cho bất cứ bài toán nào mà trong đó một tri thức về các lời giải tốt nào đó của các bài toán con hữu ích cho việc xây dựng lời giải toàn cục, tuy vậy họ đã áp dụng kỹ thuật này cho bài toán TSP hình học. Nhiều phương pháp *cắt hai* có thể được xét; những phương pháp này cắt hình chữ nhật có *n* thành phố thành hai hình chữ nhật nhỏ hơn. Thí dụ, một trong những phương pháp này chia bài toán bằng cách cắt đôi diện tích hình chữ nhật, song song với cạnh ngắn hơn; phương pháp kia giao thành phố gần giá trị *int*(n/2) nhất, với cạnh ngắn hơn của hình chữ nhật, như vậy, sẽ cung cấp một thành phố "dùng chung" giữa hai hình chữ nhật–con). Những bài toán con cuối cùng sẽ thật nhỏ (tiêu biểu là giữa 5 và 8 thành phố), và tương đối dễ giải (phương pháp 2–opt được chọn cho trường hợp này vì nó nhanh và đơn giản). Thuật giải *đắp vá* thay một số cạnh trong hai hành trình riêng biệt để có một hành trình lớn hơn. Giờ đây, vai trò chính yếu của thuật giải di truyền là quyết định hướng chia đôi (theo chiều ngang hay chiều dọc) được sử dụng tại mỗi giai đoạn. Một nhận xét thú vị là cấu trúc dữ liệu được dùng cho biểu diễn về nhiễm sắc thể của các cá thể là một mảng nhị phân

$M_{p \times p}$ tương quan với các vùng hình học của hình vuông TSP. Nếu ở một giai đoạn nào đó của thuật giải chia để trị, mà một hình chữ nhật cần được chia đôi, thì một bit được chọn từ ma trận M tương ứng gần nhất với trung tâm hình chữ nhật; giá trị của bit này xác định hướng cắt hiện hành (ngang với dọc). các toán tử di truyền được dùng trong rất đơn giản : lai hoán vị các phần tử nhị phân giữa hai mảng được cắt (trong hai điểm) dọc theo các trục x hay y. Đột biến bật tất cả các bit của mảng với xác suất không đổi (người ta dùng 0.1, nghĩa là 10% các bit trong mảng được đột biến).

Dường như một chương trình tiến hóa tốt cho bài toán TSP cần kết hợp với các toán tử cải thiện cục bộ (nhóm đột biến, dựa trên các thuật giải dùng cho tối ưu hóa cục bộ, cùng (các) toán tử nhị phân được thiết kế cẩn thận (nhóm lai tạo), sẽ kết hợp thông tin heuristic về bài toán. Ta kết thúc chương này bằng một quan sát đơn giản: yêu cầu của chương trình tiến hóa đối với bài toán TSP, có thể bao gồm biểu diễn 'tốt nhất' và các toán tử 'di truyền' để thực hiện trên chúng, vẫn còn tiếp tục!

Chương 9

CÁC BÀI TOÁN TỐI ƯU TỔ HỢP KHÁC

Như đã lưu ý trước, dường như đa số các nhà nghiên cứu đã biến đổi các cách cài đặt của họ về thuật giải di truyền hoặc bằng cách dùng biểu diễn nhiễm sắc thể không-chuẩn và/hoặc bằng cách thiết kế các toán tử di truyền chuyên biệt cho thích hợp với bài toán cần giải, để có được các chương trình tiến hóa hiệu quả. Những biến đổi như thế đã được thảo luận trong hai chương trước, dành cho bài toán vận tải và bài toán người du lịch. Trong chương này, ta khảo sát một số chương trình tiến hóa khác được các nhà nghiên cứu cài đặt, các chương trình này có biểu diễn nhiễm sắc thể không-chuẩn và/hoặc các toán tử sử dụng tri thức của bài toán. Trước hết, ta sẽ thảo luận về bài toán lập lịch, sau đó là bài toán thời khoá biểu trường học, một số bài toán phân hoạch, và cuối cùng là bài toán lập kế hoạch đi đường cho rôbô. Cuối cùng, ta sẽ kết thúc với một số chú thích ngắn gọn về một số ít bài toán thú vị khác.

Các hệ thống được mô tả và các kết quả của việc ứng dụng chúng cung cấp các luận cứ về cách tiếp cận lập trình tiến hóa, thúc đẩy việc sáng tạo các cấu trúc dữ liệu cùng với các toán tử cho phù hợp với các bài toán cần giải.

9.1. Bài toán lập lịch

Lập lịch là bài toán tổ chức sản xuất. Cần sản xuất nhiều (bộ phận) hàng hóa khác nhau; những bộ phận này có thể được xử lý theo một số kế hoạch khác nhau. Mỗi kế hoạch xử lý gồm một chuỗi thao tác; những thao tác này cần sử dụng một số tài nguyên và cần một số thời gian chạy máy. Một lịch sản xuất lập kế hoạch thực hiện các đơn đặt hàng. Trong đó, một số đơn đặt hàng có thể được thực hiện với cùng những thao tác tương đương. Nhiệm vụ là lên kế

hoạch, lập lịch sản xuất, để thực hiện các đơn đặt hàng này nhanh nhất có thể.

Bài toán lập thời lịch là chọn một chuỗi các thao tác đồng thời chỉ định về thời gian bắt đầu/kết thúc và các tài nguyên cần thiết cho mỗi thao tác. Điều cần quan tâm chính yếu là chi phí thời gian máy rỗi và năng lực lao động, và các đơn đặt hàng cần hoàn thành đúng hạn.

Có nhiều phiên bản khác nhau về bài toán lập lịch, mỗi bài được đặc trưng bằng một số ràng buộc (nghĩa là, bảo trì, thời gian lắp đặt, vv...).

Sau đây là một ví dụ đơn giản về bài toán lập lịch.

Thí dụ 9.1. Giả sử có 3 đơn đặt hàng, o_1, o_2, o_3. Đối với mỗi đơn đặt hàng, các bộ phận và số mặt hàng cần sản xuất là:

o_1: 30 × bộ phận a;

o_2: 45 × bộ phận b;

o_3: 50 × bộ phận a.

– Mỗi bộ phận có một hay nhiều kế hoạch xử lý khác nhau:

a: kế hoạch # 1_a (*thaotác*$_7$, *thaotác*$_9$);

a: kế hoạch # 2_a (thaotác$_1$, thaotác$_3$, thaotác$_7$, thaotác$_8$);

a: kế hoạch # 3_a (*thaotác*$_5$, *thaotác*$_6$);

b: kế hoạch # 1_b (thaotác$_2$, thaotác$_6$, thaotác$_7$);

b: kế hoạch # 2_b (thaotác$_1$, thaotác$_9$);

trong đó, các *thaotác*$_i$ biểu thị các thao tác cần thực hiện. Mỗi thao tác đòi hỏi một số thời gian trên một hay nhiều máy; cụ thể:

thaotác$_1$: (m_1 10) (m_3 20);

thaotác$_2$: (m_2 20);

thaotác$_3$: (m_2 20) (m_3 30);

thaotác$_4$: (m_1 10) (m_2 30) (m_3 20);

thaotác$_5$: (m_1 10) (m_3 30);

thaotác$_6$: (m_1 40);

thaotác$_7$: (m_3 20);

thaotác$_8$: (m_1 50) (m_2 30) (m_3 10);

thaotác$_9$: (m_2 20) (m_3 40);

Cuối cùng mỗi máy cần có thời gian khởi động khi thực hiện:

m_1 : 3;

m_2 : 5;

m_3 : 7.

Bài toán lập lịch có thể sử dụng GA. Davis là một trong những người đầu tiên giải bài toán này bằng GA. Ý tưởng chính trong phương pháp của ông mã hóa biểu diễn của lịch phân công là (1) các toán tử di truyền phải thực hiện theo cách có ý nghĩa, và (2) một bộ giải mã phải luôn luôn tạo ra một lời giải hợp lệ cho bài toán. Chiến lược này, mã hóa các lời giải cho các phép toán di truyền thực hiện

và giải mã chúng khi lượng giá, là hoàn toàn tổng quát và được áp dụng cho nhiều loại bài toán có ràng buộc khác. Jones cũng sử dụng những ý tưởng tương tự để giải bài toán phân hoạch.

Nói chung, ta thích một biểu diễn thông tin về các lịch phân công, như, "máy m_2 thực hiện thao tác o_i trên bộ phận a từ thời gian t_1 đến t_2". Nhưng, hầu hết các toán tử (đột biến, lai) được áp dụng vào một thông điệp như vậy có thể tạo ra lịch phân công bất hợp lệ – vì thế mà Davis đã phải sử dụng chiến lược mã hóa/ giải mã.

Ta hãy xem chiến lược giải mã được áp dụng vào bài toán lập lịch ra sao. Hệ thống của Davis duy trì một danh sách ưu tiên cho mỗi máy; những ưu tiên này được liên kết với thời gian. Phần tử đầu của danh sách là thời điểm danh sách có hiệu lực, phần còn lại của danh sách được tạo từ một số hoán vị của các đơn đặt hàng, với hai phần tử bổ sung: 'chờ' và 'rỗi'. Thủ tục giải mã mô phỏng các thao tác của công việc theo cách mà khi một máy tính sẵn sàng chọn lựa, thì thao tác cho phép đầu tiên từ danh sách ưu tiên được lấy ra. Như vậy nếu danh sách ưu tiên của máy m_1 là:

m_1: (40 o_3 o_1 o_2 'chờ' 'nhàn rỗi'),

thì thủ tục giải mã vào thời điểm 40 có thể thực hiện đơn đặt hàng o_3 trên máy m_1. Nếu không được, thủ tục giải mã sẽ thực hiện đơn đặt hàng o_1 và o_2 (nghĩa là, tìm ở o_1 trước; nếu không được mới tìm ở o_2). Biểu diễn này bảo đảm tạo một lịch phân công hợp lệ.

Các toán tử có tính chuyên biệt:

Chạy–rỗi: toán tử này chỉ được áp dụng cho những danh sách ưu tiên của các máy đã đợi hơn một tiếng đồng hồ. Nó chèn 'rỗi' làm phần tử thứ hai của danh sách ưu tiên và thiết lập lại phần tử đầu tiên (thời gian) của danh sách ưu tiên là 60 (phút).

Tranh giành: toán tử này "giành" các phần tử của danh sách ưu tiên;

Lai: toán tử này trao đổi các danh sách ưu tiên của các máy được chọn.

Xác suất thực hiện những phép toán này thay đổi từ 5% và 40% lần lượt cho **tranh giành** và **lai** lúc bắt đầu chạy, và giảm dần xuống còn 1% và 5%. Xác suất thi hành **chạy–rỗi** là phần trăm thời gian máy chờ, chia cho tổng số thời gian mô phỏng.

Tuy nhiên, các thử nghiệm chỉ được thực hiện trên một thí dụ nhỏ có hai đơn đặt hàng, 6 máy và 3 toán tử, vì vậy khó mà lượng giá được mức độ hữu dụng của phương pháp này.

Một nhóm các nhà nghiên cứu khác đã tính gần đúng bài toán lập lịch theo cách giải của TSP. Xuất phát từ nhận xét là hầu hết các toán tử được phát triển cho TSP là 'mù quáng', nghĩa là chúng không sử dụng thông tin nào về các khoảng cách thực sự giữa các thành phố. Điều này có nghĩa là những toán tử này có thể sử dụng được cho bài toán lập lịch, mà ở đó không có khoảng cách giữa hai điểm (thành phố, đơn đặt hàng, công việc, vv...). Tuy vậy, cả hai bài toán, TSP và bài toán lập lịch có những đặc trưng khác nhau. Đối với bài toán TSP, thông tin quan trọng là thông tin kề về các thành phố, trong khi bài toán lập lịch lại là thứ tự tương đối của các đơn đặt hàng. Thông tin kề không có ích cho bài toán lập lịch, còn thứ tự tương đối lại không quan trọng đối với bài toán TSP do bản chất chu trình của các hành trình: hành trình (1 2 3 4 5 6 7 8) và (4 5 6 7 8 1 2 3) thực ra là giống nhau. Vì vậy mà ta cần những toán tử khác cho nhiều ứng dụng khác. Như ta quan sát sau,

"Gil Syswerda chỉ đạo một nghiên cứu trong đó việc 'tái kết hợp cạnh' (toán tử di truyền được thiết kế đặc biệt cho TSP) thực hiện kém có liên quan với các toán tử khác trong tác vụ lập lịch. Trong

khi kích thước quần thể được Syswerda sử dụng nhỏ (30 chuỗi) và các kết quả tốt thu được từ bài toán này chỉ sử dụng đột biến (không tái kết hợp), thảo luận của Syswerda về tầm quan trọng tương đối của vị trí, thứ tự, tính chất kề đối với các tác vụ dẫn đến một vấn đề chưa được nói đến một cách đầy đủ. Các nhà nghiên cứu kể, dường như cứ ngầm cho rằng tất cả các tác vụ lập trình tự đều như nhau và chỉ một toán tử di truyền là đủ dùng cho tất cả các loại bài toán lập tuần tự."

Trước đó, Fox và McMahol cũng thực hiện một quan sát tương tự:

"Một điều quan trọng đáng để ý là khả năng áp dụng của mỗi toán tử di truyền vào nhiều loại bài toán tuần tự khác nhau. Thí dụ, trong TSP, giá trị của trình tự thì tương đương với giá trị trình tự đó theo thứ tự ngược lại. Trong bài toán lập thời biểu, đây là một lỗi lớn".

6 toán tử lập trình tự (lai thứ tự, lai ánh xạ riêng phần, tái kết hợp cạnh có tăng cường, lai trên thứ tự và lai trên vị trí – tất cả các toán tử này đã được bàn trong chương trước) được so sánh với với hai tác vụ lập trình tự khác: TSP 30–thành phố (mù quáng) và tác vụ lập trình tự 195 phần tử cho ứng dụng lập lịch. Đúng như mong đợi, các kết quả của việc tối ưu hóa thời lịch phân công (tới mức 'tốt' của 6 toán tử được nói đến) hầu như đối lập với các kết quả thu được từ TSP. Trong trường hợp tối ưu hóa lịch phân công, toán tử tái kết hợp cạnh có tăng cường là tốt nhất, tiếp theo là lai thứ tự, lai trên thứ tự và lai trên vị trí, còn PMX và lai chu trình lại xấu nhất. Ngược lại, trong trường hợp TSP, toán tử tốt nhất lại là lai trên vị trí và lai trên thứ tự, kế đó là lai chu trình và PMX, còn lai thứ tự và, tái kết hợp cạnh có tăng cường lại rất xấu. Những khác biệt này có thể giải thích bằng cách khảo sát cách các toán tử này bảo toàn thông tin về tính kề (đối với TSP) và tính thứ tự (đối với bài toán lập thời lịch).

Những quan sát tương tự cũng có thể được thực hiện cho những bài toán lập trình tự (xếp thứ tự khác). Davis mô tả một thuật giải di truyền dựa trên thứ tự cho bài toán tô màu đồ thị sau đây:

Cho một đồ thị có 8 nút có trọng và *n* màu, tô màu các nút sao cho không có hai nút kề (bằng một nối trực tiếp) nhau nào trùng màu; điểm là tổng trọng của các nút được tô màu.

Một thuật giải tham lam đơn giản sẽ sắp xếp các nút theo thứ tự giảm trọng rồi xử lý (nghĩa là, gán màu hợp lệ đầu tiên trong danh sách màu cho một nút) theo thứ tự đó. Rõ ràng, đây là bài toán lập trình tự – tối thiểu một hoán vị nút sẽ trả về lợi tức lớn nhất, vì thế ta tìm trình tự tối ưu của các nút. Cũng hiển nhiên là thuật giải tham lam đơn giản không bảo đảm lời giải tối ưu nhất: nên sử dụng một số kỹ thuật khác. Lần nữa, xét một cách phiến diện thì bài toán giống như TSP, ở đây ta theo đuổi thứ tự tốt nhất của các thành phố được người du lịch thăm. Nhưng 'bản chất' của bài toán thì khác hẳn. Thí dụ, trong bài toán tô màu đồ thị có các trọng của các nút, còn trong TSP các trọng được phân phối giữa các nút (các khoảng cách). Davis biểu diễn việc xếp thứ tự là danh sách các nút, (như (2 4 7 1 4 8 3 5 9), là biểu diễn đường dẫn của TSP) và sử dụng hai toán tử: lai trên thứ tự (được bàn trong chương 8 là một toán tử dùng cho TSP) và đột biến danh sách con tranh giành. Ngay cả đột biến có thể thực hiện một hiệu chỉnh cục bộ cho một nhiễm sắc thể, dường như cũng độc lập bài toán.

"Thường có khuynh hướng xem các đột biến là hoán vị của các giá trị thuộc hai vùng trên nhiễm sắc thể. Tôi đã thử điều này trên nhiều bài toán khác nhau, nhưng nó cũng không thể thực hiện như một toán tử mà tôi gọi là đột biến danh sách con tranh giành."

Đột biến danh sách con tranh giành chọn một danh sách con các nút từ một cha–mẹ và giành nó trong đứa con, nghĩa là cha–mẹ

(được đánh dấu bằng '|', ở chỗ bắt đầu và kết thúc của danh sách con được chọn):

p = (2 4|7 1 4 8|3 5 9)

có thể sinh con:

o = (2 4|4 8 1 7|3 5 9).

Tuy vậy, vẫn còn phải xem cách toán tử này thực hiện đối với những bài toán xếp thứ tự hay lập lịch khác.

"Nhiều loại đột biến khác có thể được dùng trong những bài toán dựa trên thứ tự. Đột biến danh sách con tranh giành là loại thường được dùng nhất. Cho đến nay chưa có báo cáo đầy đủ nào về những loại toán tử này, mặc dù đây là chủ đề đầy hứa hẹn cho sự nghiệp mai sau."

Trở về với bài toán lập lịch. Như đã nói trước đây, Syswerda xây dựng một chương trình tiến hóa cho bài toán lập lịch. Nhưng một biểu diễn nhiễm sắc thể đơn giản đã được chọn,

"Chúng tôi chọn hai phần tử cơ bản trong việc chọn một biểu diễn nhiễm sắc thể cho bài toán lập lịch. Đầu tiên là danh sách các tác vụ cần lập lịch. Danh sách này rất giống với danh sách các thành phố được thăm trong bài toán TSP. Một cách khác là trực tiếp dùng lịch phân công làm nhiễm sắc thể. Điều này có vẻ là một biểu diễn rườm rà, cần đến những toán tử phức tạp, nhưng nó có một thuận lợi nhất định khi xử lý các bài toán thực tế phức tạp như lập lịch. Trong trường hợp ở đây, biểu diễn nhiễm sắc thể trong sáng và đơn giản có ưu thế hơn biểu diễn phức tạp. Cú pháp nhiễm sắc thể mà ta dùng cho bài toán lập thời lịch là những gì đã được mô tả bên trên cho bài toán TSP, nhưng thay vì các thành phố thì ta dùng việc xếp thứ tự các tác vụ."

Nhiễm sắc thể được thông dịch bởi người lập lịch – một chương trình con 'hiểu' các chi tiết của các tác vụ lập lịch. Biểu diễn này được hỗ trợ bởi các toán tử chuyên biệt. Có 3 đột biến được xét đến: đột biến trên vị trí (hai tác vụ được chọn ngẫu nhiên, và tác vụ thứ hai được đặt trước tác vụ đầu tiên), đột biến trên thứ tự (hai tác vụ được chọn ngẫu nhiên, được hoán vị), và đột biến tranh giành (giống như đột biến danh sách con tranh giành của Davis được mô tả ở đoạn trên). Cả ba đột biến này đều thực hiện tốt hơn tìm kiếm ngẫu nhiên, với đột biến trên thứ tự rõ ràng là tốt nhất. Như đã đề cập trước đây, các toán tử lai tốt nhất đối với bài toán lập lịch là các phép lai trên thứ tự và trên vị trí.

Nhưng dường như việc chọn một biểu diễn đơn giản không hẳn là hay nhất. Phán đoán từ những thử nghiệm (không liên quan nhau) khác, như bài toán vận tải, ta thấy rằng biểu diễn nhiễm sắc thể được sử dụng phải gần gũi hơn với bài toán lập lịch. Đúng là trong những trường hợp đó cần phải thêm nỗ lực đặc biệt khi thiết kế các toán tử 'di truyền' chuyên biệt; nhưng nỗ lực này được trả công xứng đáng bằng sự gia tăng tốc độ và hiệu quả của hệ thống. Hơn nữa, một số toán tử lại không hoàn toàn đơn giản như vậy,

"Một thuật giải tham lam đơn giản chạy trên lịch phân công có thể tìm được một chỗ cho tác vụ có độ ưu tiên cao, bằng cách bỏ đi một hoặc hai tác vụ có độ ưu tiên thấp và thay chúng bằng tác vụ có độ ưu tiên cao."

Ta tin rằng, nói chung, và riêng cho các bài toán lập lịch, đây là hướng cần theo: kết hợp tri thức bài toán không chỉ vào các toán tử (như đã làm đối với biểu diễn nhiễm sắc thể đơn giản), mà phải vào cả các cấu trúc nhiễm sắc thể.

Husband, Mill và Warrington đã biểu diễn một nhiễm sắc thể là một trình tự.

$(opt_1\ m_1\ s_1)\ (opt_2\ m_2\ s_2)\ (con_3\ m_3\ s_3)$ ...,

trong đó, opr_i, m_i và s_i lần lượt biểu thị: thao tác, máy, và thời gian khởi động thứ i.

Một số tác giả khác đã so sánh ba biểu diễn, từ biểu diễn đơn giản nhất (biểu diễn – 1)

$(o_1)\ (o_2)\ (o_3)$ . . .,

đến phức tạp hơn (biểu diễn –2):

(o_1 kếhoạch # 1_a) (o_2 kếhoạch # 2_b) (o_3 kếhoạch # 2_a)... ,

và phức tạp nhất (biểu diễn –3):

$(o_1 < con_2 : m_2, con_7 : m_3, con_9 : m_2>)\ (o_2 < con_1: m_3, con_9 : m_2>)$

$(o_3 < con_1 : m_1, con_3 : m_2, con_7 : m_3, con_8 : m_1>)$...

Kết quả là biểu diễn – 3 tốt hơn hai biểu diễn kia rất nhiều.

"Chính các toán tử phải được điều chỉnh để phù hợp với các yêu cầu về miền. Biểu diễn nhiễm sắc thể nên chứa tất cả thông tin liên quan đến bài toán tối ưu hóa."

Tóm lại, có thể phân loại tất cả những phương pháp dựa trên GA thành nhiều bài toán lập lịch trên cơ sở biểu diễn nhiễm sắc thể. Và như vậy, có hai loại:

- **Biểu diễn gián tiếp**, ở đây việc biến đổi một biểu diễn nhiễm sắc thể thành một lịch sản xuất hợp lệ phải được thực hiện bởi một bộ giải mã đặc biệt (bộ lập lịch); chỉ khi đó mới có thể lượng giá một cá thể lời giải. Hơn nữa, những biểu diễn này có thể được chia thành những biểu diễn độc lập miền và những biểu diễn theo bài toán; trước đây ta đã gặp cả hai trường hợp này.
- **Biểu diễn trực tiếp,** ở đây chính lịch sản xuất được sử dụng làm nhiễm sắc thể. Biểu diễn này thường cần một số toán tử đặc biệt.

9.2. Lập thời khoá biểu cho trường học

Một trong những bài toán thú vị nhất cho nghiên cứu về các phép toán di truyền là bài toán thời khóa biểu. Bài toán này có những ứng dụng thực hành quan trọng: nó được được xem là NP–khó.

Bài toán thời khóa biểu kết hợp nhiều ràng buộc không tầm thường thuộc nhiều loại. Có nhiều phiên bản của bài toán thời biểu, một trong những bài toán này có thể được mô tả như sau:

- Có một danh sách các giáo viên $\{GV_1,...,GV_m\}$
- Một danh sách các quãng thời gian $\{T_1,... ,T_n\}$,
- Một danh sách các lớp $\{L_1,... , L_k\}$

Bài toán cần tìm thời khóa biểu tối ưu (giáo viên – thời gian – lớp); hàm mục tiêu phải thỏa những mục tiêu này (các ràng buộc mềm). Gồm có các mục tiêu sư phạm (như trải một số lớp ra nguyên tuần) những mục tiêu thuộc cá nhân (những giáo viên hợp đồng không phải dạy buổi chiều), và các mục tiêu về tổ chức (như mỗi giờ có một giáo viên bổ sung sẵn sàng cho chỗ dạy tạm thời).

Các ràng buộc gồm:

- Có một số giờ được xác định trước cho mỗi giáo viên và mỗi lớp: một thời khóa biểu hợp lệ phải "phù hợp" với những con số này,

- Chỉ một giáo viên trong một lớp vào một giờ nhất định,
- Một giáo viên không thể dạy hai lớp cùng lúc,
- Đối với mỗi lớp được xếp thời khóa biểu vào một quãng thời gian , phải có một giáo viên.

Dường như, biểu diễn nhiễm sắc thể tự nhiên nhất cho một lời giải của bài toán thời khóa biểu là biểu diễn ma trận: một ma trận $(R)_{ij}$ $(1 \le i \le m$ và $1 \le j \le n)$, ở đây mỗi hàng tương ứng với một giáo viên, mỗi cột tương ứng với một giờ, các phần tử của ma trận R là các lớp $(r_{ij} \in \{C_1,..., C_k\})$

Các ràng buộc chủ yếu được xử lý bởi các toán tử di truyền (tác giả cũng sử dụng một thuật giải sửa chữa để loại bỏ những trường hợp mà có nhiều hơn một giáo viên xuất hiện trong cùng một lớp vào cùng một giờ). Các toán tử di truyền sau đây được sử dụng,

Đột biến bậc *k*: toán tử này chọn hai chuỗi k phần tử kế nhau, từ cùng một hàng trong ma trận R, rồi hoán vị chúng,

Đột biến ngày: toán tử này là một trường hợp đặc biệt của toán tử trước: nó chọn hai nhóm cột (giờ của ma trận R tương ứng với những ngày khác nhau, rồi hoán vị chúng)

Lai: cho hai ma trận R_1 và R_2, toán tử này sắp xếp các hàng của ma trận thứ nhất theo giá trị giảm của hàm thích nghi cục bộ (một phần của hàm thích nghi chỉ thuộc về các đặc trưng cụ thể của từng giáo viên) và b hàng tốt nhất (b là tham số hệ thống, xác định trên cơ sở của hàm thích nghi cục bộ và cả hai cha–mẹ) được lấy làm khối kiến trúc; còn lại $m-b$ hàng được lấy ra khỏi ma trận trận R_2.

Chương trình tiến hóa này đã được thử nghiệm thành công trên dữ liệu của một trường lớn tại Milan, Ý.

Bài toán thời khóa biểu cũng được Paechter, Luchian và Petriuc nghiên cứu, họ đã so sánh hai phương pháp tiến hóa (phương pháp hoán vị khoảng –thời gian và phương pháp đặt và dời chỗ) đối với một bài toán thời khóa biểu thực tế, lớn. Mới đây, Burke và cộng sự mô tả thuật giải di truyền ngẫu nhiên dùng trong các bài toán lập thời khóa biểu có ràng buộc cao. Cách tiếp cận này kết hợp biểu diễn trực tiếp thời khóa biểu bằng một số ít toán tử lai heuristic và toán tử đột biến heuristic. Theo đó, thuật giải này duy trì tính khả thi của các lời giải bằng các toán tử và các cấu trúc dữ liệu chuyên biệt hóa.

9.3. Phân hoạch đối tượng và đồ thị

Bài toán phân hoạch là cần chia n đối tượng thành k loại. Lớp bài toán này gồm nhiều bài toán nổi tiếng như bài toán đóng thùng (gán các mặt hàng vào các thùng), bài toán tô màu đồ thị (gán các nút của đồ thị vào các màu cụ thể), vv... Nhiều hệ thống khác nhau đã được phát triển cho nhiều loại bài toán phân hoạch; trong phần này, ta bàn về một số bài toán đó.

Một trong những chương trình tiến hóa dựa trên việc biểu diễn tất cả các đối tượng (như các mặt hàng trong bài toán đóng thùng, hay các nút trong bài toán tô màu đồ thị là một danh sách hoán vị); có thể áp dụng những toán tử đặc biệt, và một bộ giải mã thực hiện việc quyết định của công tác này. Thí dụ, đối với bài toán tô màu đồ thị, Davis biểu diễn một hoán vị các nút trong một nhiễm sắc thể và áp dụng những toán tử chuyên biệt (lai trên thứ tự đều, đột biến trên thứ tự) cho cấu trúc này. Đồng thời, một bộ giải mã theo nguyên lý tham lam được dùng để thông dịch cấu trúc này, cách tiến hành như sau: xét một màu cụ thể và tô (nếu có thể) tất cả các nút (theo thứ tự đã được cho trong nhiễm sắc thể) bằng màu này. Khi không còn có thể tô được nữa, chuyển sang màu kế tiếp. Davis đã thực nghiệm chương trình này cho một bài toán tô màu đồ thị có 100 nút.

Von Laszewski mã hóa việc phân hoạch bằng việc mã hóa số – nhóm , nghĩa là các việc phân hoạch được biểu diễn làm *n* chuỗi các số nguyên:

$(i_1, \ldots, i_n)$,

trong đó, số nguyên thứ j , $i_j \in \{1, \ldots, k\}$ biểu thị số nhóm được gán cho đối tượng j. Nhưng, biểu diễn này được hỗ trợ "các toán tử cấu trúc thông minh": lai cấu trúc và đột biến cấu trúc.

Lai tạo cấu trúc. Cơ chế của lai cấu trúc được trình bày qua thí dụ sau.

Giả sử có hai cha–mẹ được chọn (các chuỗi –12)

p_1 = (112311232233) và p_2 = (112123122333)

Các chuỗi này được giải mã thành các phân hoạch sau:

p_1 : {1, 2, 5, 6}, {3, 7, 9, 10}, {4,8,11,12} và

p_2 : {1, 2, 4, 7} {3, 5, 8, 9},{6, 10, 11, 12}.

Trước tiên, chọn ngẫu nhiên một phân hoạch: ví dụ phân hoạch #2. Phân hoạch này được sao chép từ p_1 thành p_2:

p_2'= (112123222233)

Tiến trình sao chép này như đã thấy trong thí dụ trước thường làm hỏng yêu cầu về kích thước phân hoạch đều, vì vậy ta áp dụng một thuật giải sửa chữa. Chú ý rằng trong p_2 gốc có những phần tử được gán cho phân hoạch #2, mà không là các phần tử của phân hoạch được sao chép: đó là các phần tử 5 và 8. Những phần tử này bị xóa,

p_2''= (1121*32*2233),

và được thay thế (ngẫu nhiên) bằng các số của những phân hoạch khác, chúng được viết chồng lên trong bước sao chép. Như vậy con cuối cùng sẽ là:

p_2'''= (112133212233),

Như đã nói trước đây, trong phần mã hóa nhóm – số, hai phân hoạch giống nhau có thể được biểu diễn bởi các chuỗi khác nhau do việc đánh số phân hoạch khác nhau. Để xử lý điều này, trước khi thực hiện phép lai, việc mã hóa được thay đổi phù hợp để giảm thiểu những khác biệt giữa hai cha–mẹ.

Đột biến cấu trúc. Tiêu biểu, một đột biến có thể thay thế một thành phần của một chuỗi bằng một số ngẫu nhiên nào đó; nhưng, điều này có thể làm hỏng yêu cầu về kích thước phân hoạch đều. Đột biến cấu trúc được định nghĩa là hoán vị của hai số trong chuỗi. Như vậy, một cha–mẹ:

p = (112133212233)

có thể tạo ra con sau đây (các số ở vị trí 4 và 6 được hoán vị):

p' = (112331212233).

Thuật giải được cài đặt thành một thuật giải di truyền song song bằng các chiến lược bổ sung (như chọn lọc thay thế cha–mẹ); trên một đồ thị có 900 nút với tối đa 4 cấp độ, chương trình tiến hóa này thực hiện tốt đẹp hơn hẳn các thuật giải heuristic. Một phương pháp tương tự đã được Muhlenberg thử nghiệm với cùng cách mã hóa số – nhóm, và ông cũng sử dụng lai "thông minh", chuyển toàn bộ các phân hoạch chứ không phải các đối tượng riêng biệt.

Nhiều chương trình tiến hóa đã được John và Beltramo xây dựng cho lớp bài toán này. Những chương trình này sử dụng cách biểu diễn khác nhau và nhiều toán tử khác nhau để thao tác chúng. Khá thú vị khi quan sát tác động của việc sử dụng tri thức bài toán. đối với hiệu quả của các chương trình tiến hóa được cài đặt. Hai bài toán thử nghiệm được chọn:

- Để chia n số vào k nhóm nhằm tối thiểu hóa những khác biệt trong các tổng nhóm và;
- Phân hoạch 48 tiểu bang của Mỹ thành 4 nhóm màu để tối thiểu hóa số những cặp tiểu bang cùng biên giới vào cùng một nhóm;

Nhóm đầu tiên của các chương trình tiến hóa mã hóa các phân hoạch bằng n chuỗi các số nguyên:

$(i_1, \ldots, i_n)$,

trong đó, số nguyên thứ j, $i_j \in \{1, \ldots, k\}$ biểu thị số nhóm được gán cho đối tượng j; đây là việc mã hóa số–nhóm.

Việc mã hóa số–nhóm cho phép áp dụng các toán tử chuẩn. Một đột biến có thể thay thế một gen i_j (chọn ngẫu nhiên) bằng một số (ngẫu nhiên) trong tập $\{1, \ldots, k\}$. Các phép lai (1–điểm hay đều) luôn tạo ra con hợp lệ. Nhưng, như đã nói trong một con (sau đột biến hay lai tạo) có thể chứa ít hơn k nhóm; hơn nữa, một con của hai cha–mẹ cả hai biểu diễn cùng phân hoạch, có thể biểu diễn một phân hoạch hoàn toàn khác, do việc đánh số các nhóm khác nhau. Những thuật giải sửa chữa đặc biệt (phương pháp thải hồi, đánh số lại các cha–mẹ) đã được sử dụng để loại bỏ những rắc rối này. Cũng vậy, ta có thể xét việc áp dụng lai trên cạnh (di truyền định nghĩa trong chương 8). Ở đây ta giả sử rằng hai đối tượng nối nhau bởi một cạnh nếu và chỉ nếu chúng trong cùng nhóm. Lai trên cạnh tạo ra con mới bằng cách nối các cạnh của các cha–mẹ.

Chú ý rằng, nhiều thử nghiệm trên hai bài toán thử nghiệm này cho thấy ưu thế của toán tử lai trên cạnh; nhưng biểu diễn được sử dụng không hỗ trợ toán tử này. Thực nghiệm cho thấy nó cần gấp 2 đến 5 lần thời gian tính toán máy mà các phương pháp lai tạo khác cần đến. Đó là do biểu diễn không thích hợp: thí dụ, hai cha–mẹ:

p_1 = (11222233)

p_2 = (12222333)

biểu diễn các cạnh sau đây:

các cạnh của p_1: (12), (34), (35), (36), (45), (46), (56), (78).

các cạnh của p_2: (23), (24), (25), (34), (35), (45), (67), (68), (78).

Một con phải chứa các cạnh có trong một cha–mẹ là ít nhất, như:

(11222333)

biểu diễn các cạnh sau đây:

(12), (34), (35), (45), (67), (68), (78).

Nhưng, tiến trình chọn các cạnh lại không dễ dàng: việc chọn (5 6) và (6 7) – cả hai cạnh này đã được biểu diễn ở trên – bao hàm sự hiện diện của cạnh (5 7), không có ở đó.

Dường như, có một số biểu diễn khác thích hợp với bài toán hơn. Nhóm chương trình tiến hóa thứ hai mã hóa các phân hoạch thành các chuỗi–$(n+k-1)$ các số nguyên phân biệt,

$(i_1, \ldots, i_{n+k-1})$;

các số nguyên trong khoảng {1,..., n} biểu diễn các đối tượng, các số nguyên trong khoảng {n+1,..., n+k−1} biểu diễn các dấu phân cách; đây là một cách mã hóa *hoán vị bằng các dấu phân cách*. Thí dụ, chuỗi–7:

(1122233)

được biểu diễn là một chuỗi–9

(128345967),

trong đó, 8 và 9 là các dấu phân cách.

Dĩ nhiên, phải dùng tất cả *k-1* dấu phân cách; chúng cũng không thể xuất hiện tại vị trí đầu tiên hay cuối cùng, và cũng không thể xuất hiện liền nhau, cái này kế cái kia (nếu không, một chuỗi sẽ giải mã ra ít hơn *k* nhóm).

Như thường lệ, cần phải quan tâm đến các toán tử. Điều này có thể tương tự với một số toán tử được dùng để giải bài toán người du lịch, mà TSP được biểu diễn là hoán vị của các thành phố. Một đột biến hoán vị hai đối tượng (các dấu phân cách được bỏ đi). Hai phép lai được xét: lai thứ tự (OX) và lai phù hợp một phần (PMX) – đã được nói trong chương 8. Một phép lai có thể lặp lại thao tác của nó cho đến khi con giải mã thành một phân hoạch *k* nhóm.

Nói chung, các kết quả của các chương trình tiến hóa dựa trên hoán vị với các toán tử mã hóa sẽ tốt hơn đối với các chương trình dựa trên mã hóa số–nhóm. Nhưng, chẳng có phương pháp mã hóa nào tận dụng được nhiều tri thức bài toán. Ta có thể xây dựng một họ thứ ba của các họ tiến hóa, kết hợp tri thức vào hàm mục tiêu. Điều này được thực hiện theo cách sau.

Biểu diễn được dùng là biểu diễn đơn giản nhất: mỗi chuỗi–n biểu diễn n đối tượng:

$(i_1,..., i_n)$,

trong đó, $i_j \in \{1, . . . , k\}$ biểu thị số đối tượng, $i_j \neq i_p$ với $j \neq p$.

Thông dịch của biểu diễn này dùng một heuristic tham lam: *k* đối tượng đầu tiên trong chuỗi được dùng để khởi tạo *k* nhóm, nghĩa là, mỗi *k* đối tượng đầu tiên được đặt vào một nhóm riêng. Những đối tượng còn lại được thêm vào trên cơ sở đến–trước–đi–trước, nghĩa là, chúng được thêm vào theo thứ tự mà chúng xuất hiện trong chuỗi; chúng được đặt vào nhóm thu hoạch giá trị mục tiêu tốt nhất. Heuristic tham lam này cũng đơn giản hóa các toán tử: mỗi hoán vị mã hóa một phân hoạch hợp lệ, vì vậy ta có thể dùng lại những toán tử như trong bài toán người du lịch. Không cần phải nói, phương pháp "giải mã tham lam" có hiệu quả tốt hơn các chương trình tiến hóa dựa trên những cách giải mã khác nhiều: số–nhóm và hoán vị bằng các dấu phân cách.

Gần đây, Falkernauer đề nghị Thuật giải Di Truyền Gom Nhóm (GGA) để xử lý nhiều loại bài toán gom nhóm (phân hoạch) khác nhau; ông cố gắng chú tâm vào biểu diễn nhiễm sắc thể thích hợp để nắm được cấu trúc của bài toán. Trong phương pháp này, một nhiễm sắc thể gồm có hai phần: một *phần đối tượng* và một *phần nhóm*. Phần đối tượng dùng cách mã hóa số–nhóm: nó chứa một chuỗi *n* các số nguyên:

$(i_1,..., i_n)$,

trong đó, số nguyên thứ *j*, $i_j \in \{1, . . . , k\}$ biểu thị số nhóm được gán cho đối tượng *j*. phần nhóm của nhiễm sắc thể có chiều dài thay đổi được và chỉ biểu diễn các nhóm. Thí dụ, nhiễm sắc thể sau đây:

(1 2 1 3 3 3 1 1 2 : 1 2 3)

được thông dịch là: Phần thứ hai của nhiễm sắc thể cho thấy rằng có 3 nhóm (1, 2, và 3 – được đặc tả). Phần thứ nhất của nhiễm sắc thể

cho phép thông dịch những cấp phát: nhóm số 1 gồm các đối tượng {1, 3, 7, 8}; nhóm số 2 gồm các đối tượng {2, 9} và nhóm số 3 gồm các đối tượng {4, 5, 6}. Chú ý là ta có thể thay số '3' bằng số '5' trong biểu diễn trên (dĩ nhiên ở cả hai phần), và ý nghĩa của những cấp phát vẫn không đổi.

Ý niệm chính của biểu diễn đó là các toán tử di truyền làm việc với phần nhóm (nghĩa là, phần thứ hai) của các nhiễm sắc thể, trong khi phần thứ nhất của nhiễm sắc thể chỉ được sử dụng trong việc nhận dạng các cấp phát.

Thí dụ, đối với bài toán đóng thùng (nghĩa là, đóng thùng *n* đối tượng vào một số thùng tối thiểu có sức chứa không đổi), toán tử BPCX (Bin Packing Crossover Operator – toán tử lai tạo đóng thùng) hoạt động như sau. Giả sử hai nhiễm sắc thể cha–mẹ là:

(1 2 1 3 3 3 1 1 2 : 1 2 3) và

(2 3 3 5 1 4 2 2 6 : 2 3 4 5 6).

Hai vị trí giao nhau được chọn (ngẫu nhiên) cho mỗi phân nhóm của các nhiễm sắc thể là:

(1 2 1 3 3 3 1 1 2 : 1 | 2 3 |) và

(2 3 3 5 1 4 2 2 6 : 2 | 3 4 | 5 6).

Rồi nội dung của phần giao của cha–mẹ đầu tiên được chèn vào phần giao đầu tiên của cha–mẹ thứ hai (đối với con khác, vai trò của cha–mẹ thứ nhất và thứ hai thay đổi lại):

(... : 2_1 2_1 3_1 3_2 4_2 5_2 6_2)

Bây giờ ta có thể loại những mâu thuẫn; chú ý là nội dung của các thùng nêu trên là như sau:

thùng 2_2 – các đối tượng 1,7 và 8

thùng 2_1 – các đối tượng 2 và 9

thùng 3_1 – các đối tượng 4, 5 và 6

thùng 3_2 – các đối tượng 2 và 3

thùng 4_2 – các đối tượng 6

thùng 5_2 – các đối tượng 4

thùng 6_2 – các đối tượng 9.

Do có những đối tượng trùng lặp trong các thùng "mới" (từ cha–mẹ thứ nhất) và trong những thùng "cũ" – từ cha–mẹ thứ hai, nên ta bỏ đi những thùng "cũ" này (tạo ra những mâu thuẩn) khỏi phần thứ hai của nhiễm sắc thể, bây giờ là:

(... : 2_1 2_1 3_1)

Chú ý là ta bỏ đi thùng 6_2 (do đối tượng 9 đã có trong 2_1), thùng 5_2 (do đối tượng 4 đã có trong 3_1), thùng 3_2 (do đối tượng 2 đã có trong 2_1). Ở giai đoạn này, nhiễm sắc thể con có dạng sau:

(2_2 2_1 ? 3_1 3_1 3_1 2_2 2_2 2_1 : 2_2 2_1 3_1).

Sau khi đặt lại tên các số thùng 2_2, 2_1 và 3_1 lần lượt thành 1, 2, và 3 , nhiễm sắc thể là:

(1 2 ? 3 3 3 1 1 2 : 1 2 3).

Chú ý rằng do phương pháp của lời giải mâu thuẫn trên, đối tượng thứ ba mất đi chỗ cấp phát cho nó (điều này được đánh dấu bởi một dấu hỏi '?' trong phần đầu của nhiễm sắc thể). Như vậy ta có thể dùng một thuật giải sửa chữa heuristic nào đó để hoàn thành một cá thể hợp lệ; Falkenauer đã sử dụng heuristic giảm thích hợp đầu tiên để chèn những đối tượng còn thiếu vào. Nếu không có chỗ, cho đối tượng số 3 trong cả ba thùng trong nhiễm sắc thể trên, cần phải tạo thêm một thùng:

(1 2 4 3 3 3 1 1 2 : 1 2 3 4).

Chú ý là một lai tạo như thế có trọng trách chuyển càng nhiều thông tin có ý nghĩa từ cả hai cha–mẹ càng tốt.

Trong phương pháp trên, toán tử đột biến rất đơn giản và có ích. Nó chọn và loại bỏ một vài thùng (ngẫu nhiên). Những đối tượng không được cấp phát sẽ được chèn lại vào các thùng theo một heuristic thích hợp đầu tiên (theo thứ tự ngẫu nhiên).

Các kết quả thử nghiệm rất tốt; Falkenauer kết luận bằng nhận xét sau đây:

"Chúng tôi cũng hy vọng đã thực hiện được một trường hợp có sức thuyết phục về tầm quan trọng việc mã hóa thỏa đáng (và do đó, của các toán tử di truyền) đối với một ứng dụng thành công của hệ biến hóa GA."

9.4. Vạch lộ trình cho rôbô di chuyển

Tìm đường là một khoa học (hay một nghệ thuật) hướng dẫn lộ trình cho rôbô di chuyển qua môi trường. Vốn có trong tất cả các kế hoạch tìm đường là mong muốn đến đích mà không bị lạc hay va vào những đối tượng khác.

Thông thường, một lộ trình được lập trước để rôbô đi theo, đường đi này có thể dẫn dắt rôbô đến đích của nó, ta coi như môi trường rôbô di động đã được biết rõ hoàn toàn và không thay đổi, và rôbô có thể đi theo một cách hoàn hảo. Các bộ vạch lộ trình ban đầu là các bộ lập kế hoạch trước) hoặc chỉ thích hợp với việc vạch lộ trình trước như vậy. Tuy nhiên, các hạn chế của việc vạch lộ trình trước dẫn dắt các nhà nghiên cứu tìm hiểu việc vạch lộ trình nội tại, phụ thuộc vào tri thức thu được từ việc cảm nhận môi trường cục bộ để xử lý các chướng ngại chưa biết khi rôbô băng qua môi trường.

Chương trình tiến hóa được mô tả ở đây, nghĩa là bộ Tìm Đường Tiến Hóa (EN), đồng nhất việc vạch lộ trình trước và nội tại với một bản đồ đơn giản có độ trung thực cao và một thuật giải vạch lộ trình hiệu quả. Phần đầu của thuật giải (bộ vạch lộ trình bên ngoài) tìm lộ trình toàn cục tối ưu từ khởi điểm đến đích, trong khi phần thứ hai (bộ vạch lộ trình nội tại) có trách nhiệm xử lý những va chạm có thể xảy ra hay những vật thể chưa được biết trước bằng cách thay một phần của lộ trình toàn cục gốc bằng một hành trình con tối ưu. Cần chỉ rõ rằng cả hai phần của EN đều cùng một thuật giải tiến hóa với các giá trị tham số khác nhau. Trong những năm gần đây, nhiều nhà nghiên cứu khác đã thử nghiệm với những kỹ thuật tính toán máy tiến hóa cho bài toán vạch lộ trình. Davidor đã sử dụng cấu trúc động của nhiễm sắc thể và một toán tử lai được hiệu chỉnh để tối ưu hóa một số tiến trình thế giới thực (kể cả ứng dụng về lộ trình rôbô). Còn có một thuật giải di truyền khác dành cho bài toán vạch lộ trình, kể cả các thuật giải di truyền sử dụng chiến lược tìm kiếm đa–heuristic. Cả hai cách tiếp cận đều chấp nhận việc có bản đồ định nghĩa trước gồm có các điểm nút. Các nhà nghiên cứu khác đã sử dụng hệ thống bộ phân loại hoặc lập trình di truyền để tiếp cận bài toán vạch lộ trình. Cách tiếp cận của chúng ta là duy nhất theo nghĩa là phương pháp Tìm Đường Tiến Hóa (1) điều hành trong toàn bộ không gian trống và không tạo bất cứ một giả

định trước nào về các điểm thất nút khả thi trong lộ trình và (2) kết hợp các thuật giải vạch kế hoạch trước với kế hoạch nội tại.

Trước khi giải thích chi tiết thuật giải này, trước hết ta hãy giải thích cấu trúc bản đồ. Nhằm hỗ trợ việc tìm kiếm lộ trình trong không gian trống, liên tục và toàn vẹn, các đồ thị đỉnh được sử dụng để biểu diễn các đối tượng trong môi trường. Hiện tại, ta giới hạn môi trường là không gian hai chiều chỉ có các vật thể đa giác và chỉ có các chuyển động của rôbô là được phiên dịch. Do đó, rôbô có thể co lại thành một điểm trong khi các vật thể trong môi trường lại theo đó mà 'lớn lên'. Một rôbô di động được trang bị bằng các thụ quan siêu âm (như rôbô Denning). Một vật thể đã biết được biểu diễn bằng danh sách thứ tự (theo chiều kim đồng hồ) các đỉnh của nó. Các chướng ngại chưa biết gặp trên đường đi được mô hình hóa bằng các mảng "tường", mà mỗi mảng "tường" là một đoạn thẳng và được biểu diễn bởi danh sách các điểm nút. Biểu diễn này cũng phù hợp với biểu diễn của những vật thể đã biết, trong khi nó cũng cung cấp sự kiện là thông tin từng phần về một chướng ngại chưa biết có thể nhận được bằng cách cảm nhận tại một vị trí cụ thể. Cuối cùng, toàn bộ môi trường được định nghĩa là một vùng chữ nhật.

Bây giờ là lúc cần phải định nghĩa các lộ trình mà EN phát sinh. Một lộ trình gồm một hoặc nhiều đoạn thẳng, với vị trí khởi điểm, vị trí đích, và (có thể) các giao điểm của hai đoạn kề định nghĩa các nút. Một lộ trình hợp lệ chỉ gồm các nút hợp lệ. Một lộ trình không hợp lệ có ít nhất 1 nút không hợp lệ. Giả sử có một lộ trình $p <m_1, m_2,..., m_n>$ ($n \geq 2$), với m_1 và m_n lần lượt biểu thị nút khởi điểm và nút đích. Một nút m_i ($i = 1,..., n-1$) không hợp lệ nếu nó không thể nối với nút kế tiếp m_{i+1} do các chướng ngại vật, hay nó được định vị bên trong (hoặc quá gần với) một chướng ngại vật nào đó. Ta giả sử rằng các nút khởi điểm và đích được đặt ngoài các chướng ngại vật và cũng không quá gần chúng. Nhưng chú ý là nút khởi điểm không cần hợp lệ (nó không cần phải nối với nút kế tiếp)

trong khi nút đích phải luôn luôn hợp lệ. Cũng chú ý rằng những lộ trình khác nhau có thể có các số nút khác nhau.

Giờ ta sẵn sàng để hiểu thủ tục EN (hình 9.1)

Thủ tục tìm đường tiến hóa

Bắt đầu

Bắt đầu bộ vạch lộ trình bên ngoài

lấy bản đồ

nhận được tác vụ

thực hiện việc lập kế hoạch:

đường đi hiện hành ← FEG (xuất phát, đích)

Kết thúc bộ vạch lộ trình bên ngoài

Nếu đường đi hiện hành là hợp lệ **Thì**

Bắt đầu bộ vạch lộ trình bên trong

Lặp

di chuyển dọc theo đường đi trong khi

cảm nhận môi trường chung quanh

Nếu quá sát với một vật thể nào **Thì**

Bắt đầu

khởi điểm–cục bộ← vị thí hiện tại

đích–hiện tại←

nút gần nhất thuộc đường đi hiện tại

Nếu là vật thể mới

Thì cập nhật bản đồ đối tượng

Còn thì phát triển ảo đối tượng

tại vị trí gần nhất

thực hiện việc vạch lộ trình:

đường đi–hiện tại← NEG

(khởi điểm–hiện tại, đích–hiệntại)

cập nhật đường đi hiện tại

Hết

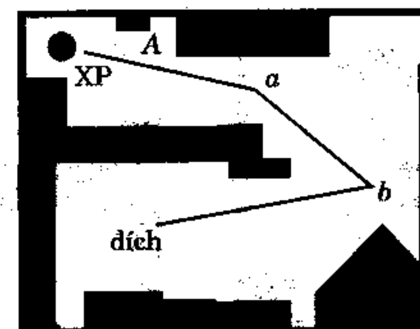
Đến khi (đến đích) **hoặc** (điều kiện thất bại)

Kết thúc bộ vạch lộ trình bên trong

Kết thúc

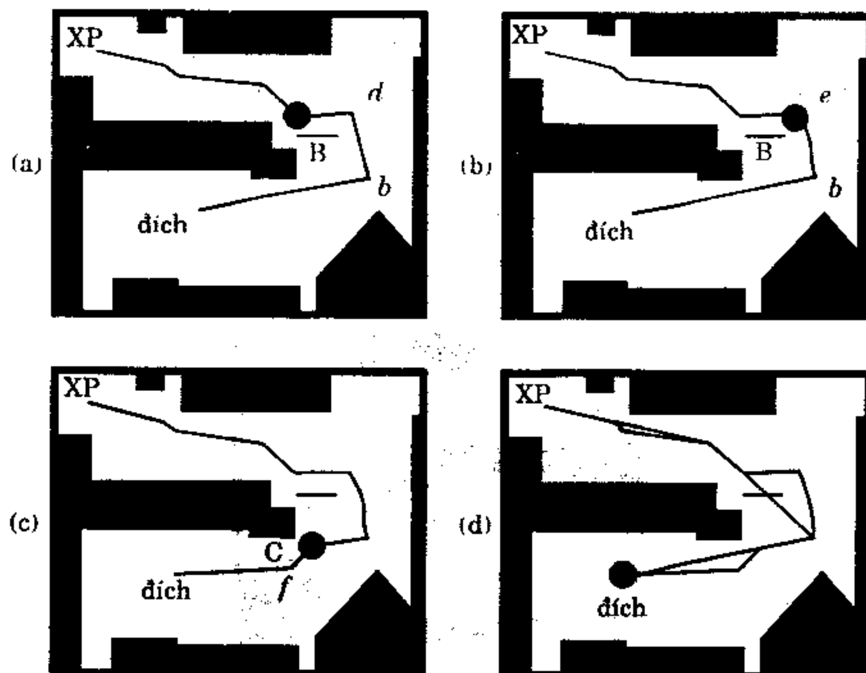
Hình 9.1. Mã giả thuật giải tiến hóa.

Trước tiên EN đọc bản đồ rồi nhận các vị trí khởi điểm và đích của tác vụ. Rồi FEG (oFf–line Evolutionary alGorithm) phát sinh một lộ trình toàn cục gần–tối ưu, một lộ trình đường từng đoạn thẳng gồm các điểm thất nút khả thi hoặc các nút. Hình 9.2 trình bày một lộ trình toàn cục như vậy được phát sinh bởi FEG. (Chu trình trọn vẹn mô phỏng rôbô).



Hình 9.2. Môi trường và lộ trình toàn cục.

Vì rôbô bắt đầu đi theo lộ trình để tiến về đích, nó cảm nhận môi trường đối với việc gần gũi những vật thể gần đó, và NEG(oN–line Evolutionary alGorithm) được sử dụng để phát sinh các lộ trình cục bộ nhằm đối phó với những vật thể và những va chạm bất ngờ. Để mô phỏng tác động của những vật thể chưa biết trong môi trường có những tập tin dữ liệu bổ sung được tạo để biểu diễn những chướng ngại vật đó (như các mảng "tường" được giải thích trước đây). Chúng tôi đã thử nghiệm với 5 tập vật thể chưa biết khác nhau; hình 9.3 a–d cho thấy hành động của rôbô đối với một trong những tập này.



Hình 9.3. Lộ trình thực tế

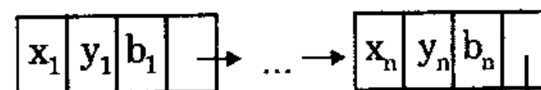
Khi rô bô di chuyển quá gần phía góc trái – dưới của vật thể ở gần 'A', NEG và phát sinh một lộ trình cục bộ để quẹo ra xa 'A', cũng là lộ trình từng đoạn thẳng. Rồi rôbô đi theo lộ trình hiện hành một cách thành công để đến điểm 'a'. Trong khi rôbô đi từ 'a' đến 'b', nó khám phá ra vật thể chưa biết hoặc vật thể 'B'. Bây giờ EN cập nhật bản đồ và lần nữa NEG phát sinh hành trình cục bộ với điểm thắt nút 'd' (hình 9.3a). Khi rôbô di chuyển từ 'd' đến 'b', nó tiến đến quá gần vật thể 'B': do đó, một hành trình cục bộ khác được phát sinh do được biểu diễn bởi điểm thắt nút 'e' (hình 9.3b). Rồi rôbô di chuyển từ 'd' đến 'e' và cuối cùng đến đích con 'b'. Bước kế tiếp là đi từ 'b' đến đích, như trong hình 9.3c, đoạn lộ trình này quá sát góc phải–dưới của vật thể 'C'. Do đó, một lộ trình cục bộ khác

được phát sinh khi được biểu diễn bởi điểm thắt nút 'f' và rồi đến 'đích'. Hình 9.3d cho thấy lộ trình toàn cục gốc và lộ trình thực sự đã đi. Chú ý là tiến trình tìm đường chấm dứt khi rôbô đến đích hoặc có một điều kiện thất bại xảy ra, nghĩa là EN không tìm được lộ trình khả thi trong một khoảng thời gian nào đó (nghĩa là, trong số thế hệ được xác định trong NEG).

Như đã nói, EN kết hợp việc vạch lộ trình nội tại với lộ trình cài trước với cùng cấu trúc dữ liệu và cùng thuật giải tìm lộ trình. Nghĩa là, khác biệt duy nhất giữa FEG và NEG là ở các tham số được dùng: kích thước quần thể *pop-size*$_g$, số thế hệ *Tg*, chiều dài tối đa của nhiễm sắc thể n_g, vv... đối với FEG, và *pop-size*$_l$, Tl, nl, vv... đối với NEG. Chú ý là cả hai FEG và NEG đều thực hiện việc vạch lộ trình toàn cục, ngay cả khi NEG thường phát sinh một hành trình cục bộ, nó thực hiện trên *bản đồ toàn cục* được cập nhật. Hơn nữa, nếu không có trong môi trường, hoặc không có vật thể nào được biết ngay từ đầu giữa vị trí khởi điểm và vị trí đích, thì FEG sẽ phát sinh một lộ trình thẳng chỉ có hai nút: vị trí khởi điểm và vị trí đích. Nó sẽ chỉ phụ thuộc vào NEG để dắt rôbô đến đích trong khi tránh những vật thể chưa biết.

Dưới đây, ta bàn chi tiết các thành phần của NEG và FEG.

Các nhiễm sắc thể là những danh sách thứ tự của các nút lộ trình như trong hình 9.4. Mỗi nút lộ trình, cách khỏi con trỏ trỏ đến nút kế, gồm có các toạ độ x và y của một điểm thắt nút trung gian dọc theo lộ trình và một biến Bool b, cho biết nút được cho là khả thi hay không.



Hình 9.4. Nhiễm sắc thể biểu diễn một lộ trình.

Chiều dài của các nhiễm sắc thể (số nút lộ trình được biểu diễn trong một nhiễm sắc thể) có thể thay đổi. Trong cách vạch lộ trình bên ngoài, chiều dài tối đa của một nhiễm sắc thể được thiết lập là số n_g của các đỉnh, biểu diễn các vật thể đã biết trong môi trường. Không phải là tất cả các lộ trình khả thi đều cần một số lớn (như: n_g) nút trung gian: ngay cả trong các môi trường phức tạp thì lộ trình khả thi cũng có thể hoàn toàn đơn giản. Do đó, ta cho chiều dài của nhiễm sắc thể biến đổi được để đối phó với những tình huống đó một cách uyển chuyển tốt đẹp.

Trong khi vạch lộ trình nội tại, lộ trình cục bộ để đi vòng qua một chướng ngại vật cũng chỉ chứa một số nhỏ các nút, do đó tham số n_l là chiều dài tối đa của nhiễm sắc thể trong giai đoạn này, tương đối nhỏ vào lúc bắt đầu tìm kiếm cục bộ. Nhưng nếu tiến trình tiến hóa không tìm được lộ trình hợp lệ sau một số thế hệ, chiều dài tối đa của nhiễm sắc thể sẽ tăng lên: trong tình huống này thì các lộ trình hợp lệ có thể có nhiều cấu trúc phức tạp hơn. Trong hệ thống EN, ta đã sử dụng tham số *nl* là hàm của số thế hệ hiện hành *t*, chính xác hơn là $n_l(t) = t$.

Các quần thể khởi tạo (*pop–size*$_g$ đối với FEG và *pop–size*$_l$ đối với NEG) của các nhiễm sắc thể được phát sinh ngẫu nhiên. Đối với mỗi nhiễm sắc thể, một số ngẫu nhiên được phát sinh trong khoảng 2.. max(2, n_g) (đối với bộ vạch lộ trình bên ngoài) để xác định chiều dài của nó. Các tọa độ *x* và *y* được tạo ngẫu nhiên cho mỗi nút của nhiễm sắc thể đó (các giá trị của tọa độ được giới hạn bên trong biên giới của môi trường).

Đối với mỗi nút của một nhiễm sắc thể, giá trị của biến Bool *b* được xác định (kiểm soát tính khả thi). Nếu nút khả thi, giá trị b của nó được thiết lập là TRUE, ngược lại sẽ là FALSE. Các phương pháp kiểm soát tính khả thi của một nút (nghĩa là sự hợp lệ của vị trí), sự không vướng các vật thể ở gần, và khả năng liên lạc) là tương đối đơn giản và dựa trên các thuật giải do Pavlidis mô tả.

Tính thích nghi (tổng chi phí lộ trình) của một nhiễm sắc thể *p* = $<m_1, m_2, ..., m_n>$ được xác định bởi hai hàm lượng giá riêng biệt (cho những cá thể khả thi và không khả thi):

- Đối với lộ trình hợp lệ *p*:

Path–Cost (p) = w_d**dist(p) +*w_s**smooth(p) +*w_c**clear(p),*

trong đó, các trọng w_d, w_s, và w_c chuẩn hóa tổng chi phí của một lộ trình, và

$$- \; dist(p) = \sum_{i=1}^{n-1} d(m_i, m_{i+1}),$$

trong đó, $d(m_i, m_i+1)$ là khoảng cách giữa các điểm thắt nút m_i và m_{i+1}; nghĩa là hàm *dist(p)* trả về tổng chiều dài của lộ trình *p*.

$$- \; smooth(p) = \max_{i=2}^{n-1} s(m_i)$$

trong đó:

$$s(m_i) = \frac{\theta_i}{\min\{d(m_{i-1}, m_i), d(m_i, m_{i+1})\}}$$

nghĩa là hàm *smooth(p)* trả về độ cong lớn nhất của *p* tại một điểm thắt nút.

$$- \; clear(p) = \max_{i=2}^{n-1} c_i$$

trong đó:

$$c_i = \begin{cases} d_i - \tau, & \text{nếu } d_i \geq \tau \\ a(\tau - d_i), & \text{nếu ngược lại} \end{cases}$$

d_i là khoảng cách tối thiểu giữa đoạn (m_i, mi+1) của lộ trình và tất cả các vật thể đã biết, τ định nghĩa khoảng cách an toàn, và *a* là hệ số; nghĩa là hàm *clear(p)* trả về con số lớn nhất để đo sự không vướng mắc giữa tất cả các đoạn của *p* và những vật thể.

- Đối với lộ trình không hợp lệ p:

Path–Cost $(p) = \alpha + \beta + \gamma$

trong đó, α là số giao điểm của lộ trình p với tất cả các 'tường' của các vật thể, β là số giao điểm trung bình của mỗi đoạn không hợp lệ, và γ là chi phí của lộ trình hợp lệ tệ nhất trong quần thể hiện hành; do biến cuối cùng này, mà tất cả các lộ trình hợp lệ trong quần thể đều tốt hơn bất cứ lộ trình không hợp lệ nào.

Nhiều toán tử (lai,hay đột biến, chèn, xóa, làm trơn, và hoán vị) cũng có trong FEG và NEG. Ta sẽ lần lượt bàn về chúng.

Lai. Lai tạo này cũng giống lai 1 điểm cổ điển được sử dụng rộng rãi trong các thuật giải di truyền. Nó tái kết hợp những phần "tốt" của các lộ trình có trong cả hai cha–mẹ, để tạo ra một lộ trình đầy hy vọng tốt hơn mà con biểu diễn. Hai nhiễm sắc thể được chọn bị cắt tại một số vị trí và được dán lại với nhau: phần đầu của nhiễm sắc thể thứ nhất với phần thứ nhì của nhiễm sắc thể thứ hai, và phần đầu của nhiễm sắc thể thứ hai với phần thứ nhì của nhiễm sắc thể thứ nhất. Nhưng các điểm giao trong cả hai nhiễm sắc thể đều không được chọn ngẫu nhiên: nếu có các nút không khả thi trong nhiễm sắc thể, các điểm giao rơi vào sau một trong các nút đó.

Đột biến–1 Đột biến này có nhiệm vụ dò tìm chính xác các giá trị tọa độ của các nút liệt kê trong nhiễm sắc thể. Nếu một nút của một nhiễm sắc thể được chọn cho đột biến này, thì các tọa độ của nó được hiệu chỉnh. Thí dụ, tọa độ $x \in \langle a, b\rangle$ (cũng như tọa độ y được thay đổi như sau:

$$x' = \begin{cases} x - \delta(t, x - a), & \text{nếu } r = 0 \\ x - \delta(t, b - x), & \text{nếu } r = 1 \end{cases}$$

với r là bit ngẫu nhiên, và hàm $\delta(t,z)$ trả về một giá trị thuộc $[0...z]$ sao cho xác suất của $\delta(t,z)$ gần bằng 0 sẽ tăng khi t tăng (t là số thế

hệ hiện hành của tiến trình tiến hóa). Toán tử được mô hình hóa trên đột biến không đồng dạng được dùng trong các hệ thống tiến hóa cho việc tối ưu hóa phi tuyến. Đột biến này có nhiệm vụ "làm trơn" dạng của lộ trình.

Đột biến_2. Đột biến này có ích trong những trường hợp cần có một thay đổi lớn hơn trong một giá trị (tình huống này thường xảy ra trong giai đoạn vạch lộ trình bên trong, khi một chướng ngại vật làm nghẽn lộ trình). Nếu một nút của nhiễm sắc thể được chọn cho đột biến này, các tọa độ của nó được hiệu chỉnh. Thí dụ, tọa độ $x \in \langle a,b\rangle$ (cũng như tọa độ y) được biến đổi như sau:

$$x' = \begin{cases} x - \Delta(t, x - a), & \text{nếu } r = 0 \\ x - \Delta(t, b - x), & \text{nếu } r = 1 \end{cases}$$

với r là bit ngẫu nhiên, và hàm $\Delta(t,z)$ trả về một giá trị trong khoảng $[0,..z]$ sao cho xác suất $\Delta(t,z)$ gần bằng z sẽ tăng khi số thế hệ t tăng.

Chèn. Toán tử này chèn một nút mới vào lộ trình đang có; chỗ nào giữa hai nút này cũng có cùng xác suất của việc chèn đó.

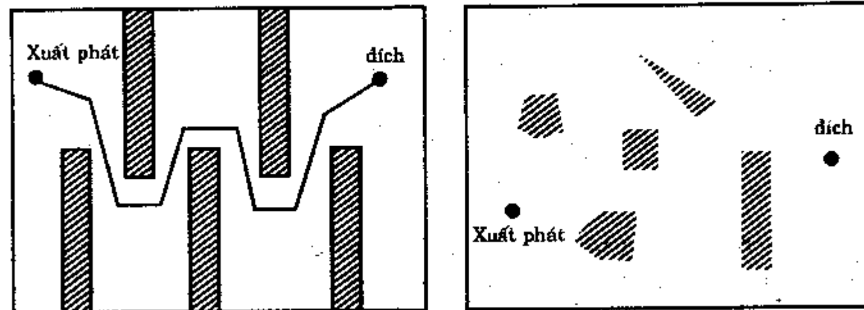
Xóa. Toán tử này xóa một nút khỏi lộ trình; nút nào cũng có cùng xác suất xóa.

Làm trơn. Toán tử này làm trơn một phần lộ trình ở chỗ quẹo đột ngột. Đối với điểm thắt nút m_i được chọn (có độ cong cao), toán tử này chọn hai điểm thắt nút mới k_1 và k_2 (lần lượt từ các đoạn (m_{i-1}, m_i) và (m_i, m_{i+1})), chèn chúng vào lộ trình, bỏ đi m_i; và nó tạo một lộ trình mới:

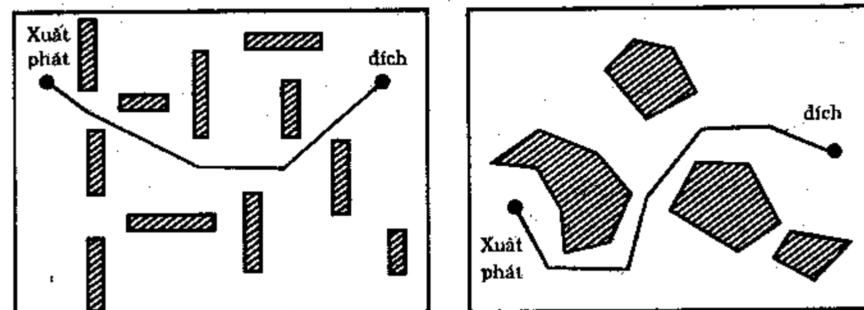
$$p' = (m_1, \ldots, m_{i-1}, k_1, k_2, m_{i+1}, \ldots, m_n)$$

Hoán vị. Toán tử này cắt nhiễm sắc thể được chọn thành hai phần (điểm cắt được xác định ngẫu nhiên) rồi hoán vị hai phần này.

Dựa trên những kết quả thử nghiệm sơ khởi, EN đã chứng tỏ là hiệu quả và thành công so với những việc tìm đường bằng những phương pháp truyền thống. Các kết quả của phiên bản hiện hành của hệ thống trên hai môi trường khác nhau được trình bày trong các hình 9.5 và 9.6.



Hình 9.5. Kết quả của EN trên hai môi trường.



Hình 9.6. Kết quả của EN trên hai môi trường khác.

Dĩ nhiên, có một nhu cầu khai thác tiềm năng của EN bằng cách thực hiện thêm nhiều thử nghiệm trong những môi trường khác

nhau, quan trọng nhất là bằng việc cài đặt EN trên một rôbô thực sự. Đồng thời, vẫn còn nhiều vấn đề về việc tìm đường tiến hóa cần được giải quyết; gồm có (1) thiết kế những điều kiện dừng thông minh hơn cho FEG và NEG để nhận thức tốt hơn các đích tối ưu hóa (hiện nay các thuật giải dừng khi tìm được một lộ trình khả thi hoặc khi đã trải qua một số thế hệ cố định) (2) đưa ra những tần suất chuyển đổi của toán tử di truyền khác với những tần suất trong phiên bản hiện hành của hệ thống (hiệu chỉnh này sẽ nâng cao hiệu quả của hệ thống và chỉ dựa trên nhận xét là những toán tử khác nhau có lẽ sẽ giữ những vai trò khác nhau, ở những giai đoạn khác nhau, trong quá trình tiến hóa) (3) mở rộng EN để thực thi trong một môi trường có những vật thể không đa diện, (4) kết hợp tri thức của giai đoạn hiện hành về việc tìm kiếm vào cách vận hành của các toán tử (như, có thể có nghĩa hơn khi băng qua hai lộ trình tại những điểm thắt nút không khả thi), và (5) khai thác một cơ chế học nào đó để EN có thể tận dụng được những kinh nghiệm đã qua.

Tuy vậy, mặc dù EN rất có hiệu quả và thành công trong nhiều trường hợp, nó vẫn có một hạn chế lớn: nó cho rằng lộ trình thực sự đủ tốt và khả thi có thể đạt được bằng sự xáo trộn nhỏ từ lộ trình tốt nhất hiện hành; hệ thống không được thiết kế để có thể thay thế lộ trình toàn cục hiện hành ; ở một giai đoạn nào đó trong khi băng qua môi trường, bằng một lộ trình toàn cục khác (có thể là tốt hơn). Như thế cũng đáng để thử nghiệm với các lời giải khác. Không như EN gồm có các bộ vạch lộ trình bên trong và bên ngoài, một bộ tìm lộ trình chuyển đổi có thể hoàn toàn thuộc loại bên trong; nó có thể không ngừng chuyển lộ trình liên kết vị trí hiện hành của rôbô và đích dựa trên thông tin cảm nhận mới thu được.

9.5. Lưu ý

Trong đoạn cuối cùng này, ta bàn ngắn gọn về một số ít các ứng dụng tương đối mới của kỹ thuật tiến hóa, do một số lý do, rất thú vị

(từ triển vọng xây dựng một chương trình tiến hóa). Ta sẽ lần lượt thảo luận chúng.

Có một số ứng dụng (như các bài toán thiết kế mạng máy tính), trong đó lời giải là đồ thị. Bài toán biểu diễn các đồ thị trong thuật giải di truyền là một bài toán khá thú vị kiểu đó. Mới đây, Palmer và Kershenbaum đã thử nghiệm theo nhiều cách biểu diễn cây. Họ nhận dạng nhiều thuộc tính cần có của một biểu diễn tốt; các thuộc tính này gồm:

1. Tính đầy đủ: khả năng biểu diễn mọi loại cây khả hữu,
2. Tính không có nhiễu (bias): tất cả các cây có thể được biểu diễn với cùng số lượng mã hóa,
3. Độ tin cậy: chỉ biểu diễn cây,
4. Tính hiệu quả: dễ dàng chuyển dạng giữa biểu diễn cây mã hóa và biểu diễn cây theo nhiều dạng qui ước thích hợp cho việc lượng giá hàm thích nghi và các ràng buộc,
5. Tính cục bộ: những thay đổi nhỏ trong biểu diễn cây tạo ra những thay đổi nhỏ trong cây.

Đương nhiên, biểu diễn lý tưởng của cây sẽ có tất cả các thuộc tính trên. Tuy vậy, đa số các biểu diễn hiện có không đạt được điều này. Ngoài ra còn một vài biểu diễn. Các biểu diễn này gồm:

- Biểu diễn vectơ đặc trưng, ở đây cây được biểu diễn bằng vectơ nhị phân có chiều dài bằng số cạnh của đồ thị cấp trên,
- Biểu diễn tiền bối (ông bà), ở đây nút được thiết kế là gốc và các ông bà của mỗi nút được ghi nhận: ở đây cây được mã hóa là vectơ số nguyên có chiều dài bằng số nút,

- Biểu diễn các số Prufer, ở đây cây được mã thành số có $n-2$ "chữ số" (là số nút của cây), mỗi "chữ số" là số nguyên, xác định bằng một thuật giải đặc biệt.

Các tác giả cũng đề nghị một biểu diễn mới, dựa trên một quan sát đơn giản là một số nút nhất định phải nằm bên trong và các nút khác phải là nút lá. Trong biểu diễn này nhiễm sắc thể giữ giá trị nhiễu của mỗi nút và mỗi cạnh có thể có (như vậy, cây được biểu diễn bằng vectơ $n+\frac{n(n+1)}{2}$ số); các nhiễu biến đổi ma trận chi phí C_{ij} của đồ thị:

$$C_{ij}' = C_{ij} + P_1(C_{max})b_{ij} + P_2(C_{max})(B_i + b_j),$$

trong đó, P_1 và P_2 là các tham số của phương pháp, và C_{max} là chi phí liên kết tối đa. Cây mà nhiễm sắc thể biểu diễn được tìm thấy bằng cách áp dụng thuật giải Prim, để tìm cây bắc cầu tối thiểu qua các nút, bằng ma trận chi phí bị nhiễu. Biểu diễn này cũng có thể mã hóa bất cứ cây nào được cung cấp các giá trị b_i thích hợp.

Một văn bản khác của Abuali và cộng sự khám phá một kế hoạch mã hóa mới để biểu diễn các cây bắc cầu (đối với bài toán cây bắc cầu tối thiểu). Kế hoạch mã hóa này dựa trên các nút được gọi là định thức, là các vectơ có $n-1$ số nguyên, số k thứ i trong một mã định thức tương ứng với một cạnh từ đỉnh k đến i+1.

Esbensen trình bày một thuật giải di truyền tìm kiếm các cây Steiner tối ưu. Bài toán cây Steiner (phiên bản thuộc loại NP–đầy đủ) được hình thức hóa như sau: cho một đồ thị và một tập con xác định các đỉnh, công việc phải làm là tìm một đồ thị con chi phí tối thiểu bắc qua các đỉnh xác định. Tác giả đã dùng một bộ giải mã tất định thông dịch các tập các đỉnh Steiner được chọn là một cây Steiner hợp lệ. Như vậy mỗi nhiễm sắc thể là một chuỗi nhị phân mà mỗi bit tương ứng với một đỉnh; một chuỗi nhị phân như thế biểu diễn một tập con các đỉnh Steiner. Tập con các đỉnh này cùng với

các đỉnh xác định tạo thành khởi điểm cho bộ giải mã, bộ này (1) xây dựng đồ thị con do các đỉnh này gây ra; (2) tính toán máy cây bắc cầu tối thiểu cho đồ thị con này; (3) xây dựng (từ cây bắc cầu tối thiểu này một đồ thị con khác bằng cách thay mỗi cạnh bằng một lộ trình ngắn nhất tương ứng trong đồ thị gốc; (4) tính toán một cây bắc cầu tối thiểu cho đồ thị con cố được; và (5) tính toán cây Steiner bằng cách xóa lần lượt (từ cây bắc cầu tối thiểu mới nhất) tất cả các đỉnh không có trong danh sách gốc các đỉnh cấp 1.

Bài toán phủ tập hợp (SCP) là bài toán phủ các hàng của một ma trận nhị phân $n \times k$, $A = (a_{ij})$, bằng một tập con các cột với chi phí tối thiểu; bài toán có nhiều ứng dụng thực hành (vị trí của các phương tiện, lập thời biểu nhóm làm việc, vv...). Mỗi cột $1 \le j \le k$ có kết hợp chi phí c_j; như vậy SCP có thể được diễn tả là:

$$\sum_{j=1}^{k} c_j x_j$$

với x_j biểu thị một biến quyết định nhị phân ($x_j = 1$ nếu và chỉ nếu cột j được chọn trong bao) thỏa:

$$\sum_{j=1}^{k} a_{ij} x_j \ge 1 \quad ,$$

với mọi $1 \le i \le n$.

Beasley thử nghiệm nhiều bài toán SCP với thuật giải di truyền được hiệu chỉnh, từ 200 hàng, 1000 cột đến 1000 hàng 10000 cột. Các kết quả thật tốt; thuật giải đã phát sinh những lời giải tối ưu cho những bài toán có kích thước nhỏ hơn và những lời giải chất lượng cao cho những bài toán kích thước lớn hơn.

Chú ý rằng SCP có một biểu diễn "lý tưởng" từ triển vọng của một GA: chuỗi nhị phân của các x_j $(1 \le j \le k)$ biểu diễn một lời giải

cho bài toán. Không cần thêm việc mã hóa nào. Hàm lượng giá (cho những cá thể khả thi) cũng dễ hiểu; chỉ là:

$$f(x) = \sum_{j=1}^{k} c_j x_j$$

Thách thức chính trong SCP là vấn đề tính khả thi; bất cứ toán tử nào được áp dụng cho các chuỗi nhị phân như thế cũng có thể tạo ra các con vi phạm ràng buộc của bài toán (nghĩa là, có một số hàng không được phủ). Beasley đã dùng một thuật giải sửa chữa để duy trì tính hợp lệ của lời giải. Thuật giải sửa chữa này có nhiệm vụ phủ những hàng chưa được phủ; việc tìm những cột bổ sung dựa trên tỉ lệ giữa chi phí của một cột và số các hàng chưa phủ mà nó sẽ phủ. Một lưu ý thú vị là một khi tiến trình sửa chữa hoàn tất và một cá thể không hợp lệ được chuyển thành hợp lệ, thì một bước tối ưu hóa cục bộ được thực hiện, trong đó ta có thể bỏ đi các cột không cần thiết (nghĩa là, các cột có thể bỏ đi mà không vi phạm các ràng buộc).

Bui và Eppley mô tả thuật giải di truyền kết hợp dùng cho bài toán "nhóm" tối đa và cung cấp những kết quả thực nghiệm của nó trên những trường hợp thử nghiệm có đến 1500 đỉnh và trên nửa triệu cạnh. Bài toán nhóm tối đa là bài toán tìm một đồ thị con đủ (nghĩa là, nhóm) của một đồ thị được cho có kích thước cực đại (được đo bằng số đỉnh). Phiên bản quyết định của bài toán là NP–đầy đủ; không có gì ngạc nhiên là nhiều phương pháp tính gần đúng khác nhau đã được đề nghị. Chú ý là công việc phải làm là tìm một tập con các đỉnh, do đó có thể dùng biểu diễn nhị phân dễ hiểu: vectơ nhị phân.

$$<b_1,\dots,b_n>$$

định nghĩa tập con đó (với mọi n đỉnh được sắp xếp theo thứ tự ngẫu nhiên, mà tất cả các đỉnh là $b_i = 1$ bao hàm việc chọn nút thứ i). Thay vì thiết kế các toán tử phức tạp có thể duy trì tính khả thi của các

lời giải (nghĩa là, có thể bảo đảm rằng con là một nhóm) hoặc thay vì thiết kế một thuật giải sửa chữa (có thể sửa một lời giải ngẫu nhiên thành một nhóm), các tác giả xây dựng một hàm mục tiêu thông minh có thể phân biệt giữa các không là–nhóm và gần như–nhóm; hàm này được định nghĩa là:

$$f(X) = \alpha|X| + \beta\frac{2e(X)}{|X|(|X|-1)}$$

với α và β là số nguyên (lần lượt được gọi là trọng lực lượng và trọng hoàn chỉnh) và *e(X)* trả về các cạnh trong đồ thị con do *X* phát sinh. Một chú ý thú vị là tỉ lệ β/α có thể thay đổi: ban đầu thì nó nhỏ (giai đoạn khai thác, mà tính cơ bản của lời giải được nhấn mạnh) và khi thuật giải tiến triển, tỉ lệ này tăng (tính hoàn chỉnh của lời giải được nhấn mạnh). Điều quan trọng cần để ý là thuật giải được mở rộng bằng cách kết hợp một thủ tục con tối ưu hóa cục bộ. Thủ tục con heuristic này (1) quyết định xem việc loại bỏ một đỉnh nào đó (các đỉnh được xét theo một thứ tự đặc biệt) có cải thiện độ thích nghi của lời giải không (nếu có, đỉnh bị loại) và (2) cũng cố gắng tăng tính cơ bản của lời giải; nó quyết định xem thêm vào một đỉnh có tăng giá trị thích nghi của lời giải không (nếu có, đỉnh được thêm vào).

Một ứng dụng thú vị của kỹ thuật tiến hóa để chất hàng lên tấm đỡ được Juliff mô tả. Bài toán này là bài toán lập thời lịch đặc biệt gồm có (1) chất các hộp chứa hàng lên các tấm đỡ và (2) chồng các tấm này lên xe tải. Yêu cầu bài toán thật phức tạp do phải duy trì một cân bằng về tải (mọi lúc) trong khi giao hàng, và cực đại hóa tính hiệu quả của việc chất lên và dỡ hàng xuống. Cũng như đối với những bài toán lập lịch, ta có thể xây dựng một hệ thống với các biểu diễn trực tiếp hoặc gián tiếp; trong trường hợp gián tiếp, bộ lập lịch (hoặc trong trường hợp này là bộ xây dựng tải), có thể hoàn thành công việc. Nhưng, cả hai cách tiếp cận này đều có những bất lợi: các toán tử bài toán đặc tả cao cấp (biểu diễn trực tiếp) hay tìm kiếm giới hạn (biểu diễn gián tiếp): thường thì một nhiễm sắc thể

biểu diễn một tính năng còn những tính năng khác không được biểu diễn đầy đủ (tri thức của những tính năng bổ sung này chỉ được kết hợp trong bộ xây dựng tải) vì thế tìm kiếm di truyền không được hướng dẫn đầy đủ.

Juliff phát triển một số hệ thống cho việc chất hàng lên tấm đỡ hàng dựa trên biểu diễn gián tiếp và những bộ xây dựng tải thông minh; một trong những hệ thống này kết hợp một cấu trúc đa nhiễm sắc thể để xử lý nhiều tính năng khác nhau của bài toán (thực ra có 3 nhiễm sắc thể được dùng); và không có gì ngạc nhiên, hệ thống đa nhiễm sắc thể thành công hơn các hệ thống một nhiễm sắc thể.

Gần đây, nhiều ứng dụng thú vị của kỹ thuật tiến hóa vận dụng vào nhiều bài toán thuộc khoa học quản lý (gồm những bài toán lập lịch và thời khóa biểu) đã được báo cáo.

Sự quan tâm vào các kỹ thuật tiến hóa của kỹ thuật công nghệ; nhiều bài viết đề cập những vấn đề cụ thể và cung cấp những mô tả về những lời giải cụ thể trong lĩnh vực này.

Ta hãy kết luận chương này bằng một nhận xét chung là ngày càng có nhiều ứng dụng khác nhau về các kỹ thuật tiến hóa cho những bài toán tối ưu tổ hợp (kể cả TSP) một số thuật giải tìm kiếm cục bộ heuristic để tăng hiệu quả. Điều này được thực hiện như một bước riêng rẽ của thuật giải, hoặc những thuật giải đó được sử dụng như những đột biến thông minh. Các kết quả do Yagiura và Ibaraki báo cáo trong nghiên cứu gần đây của họ cho thấy rằng những thuật giải tìm kiếm cục bộ di truyền như vậy là những kỹ thuật mạnh sánh ngang với những kỹ thuật khác (tìm kiếm cục bộ đa khởi điểm ngẫu nhiên hay các thuật giải di truyền thuần chất).

Những kết quả này cũng nhấn mạnh sự quan trọng của các toán tử đột biến, chúng không chỉ là các toán tử phụ, mà là những thành phần thiết yếu của tất cả các hệ thống tiến hóa. Điều này phù hợp (ở

một nghĩa nào đó) với các kết quả gần đây của Jones, ông đã thử nghiệm với các đột biến vĩ mô và cho thấy tầm quan trọng trong các cách mã hóa nhị phân (đối với toán tử lai tạo). Dù thế nào, dường như tính cơ bản của các mẫu tự được dùng cho việc mã hóa các cá thể càng cao, vai trò của (các) toán tử đột biến càng lớn. Đây là trường hợp của các thuật giải mà các cá thể được biểu diễn bằng những vectơ thực, các số nguyên, các ma trận, các máy trạng thái hữu hạn, vv... Không có gì ngạc nhiên khi có một khuynh hướng mới trong phương pháp lập trình di truyền, có thể nhấn mạnh vai trò của nhiều loại toán tử đột biến khác nhau; điều này cũng cho phép giảm rất nhiều các kích thước quần thể trong các phương pháp lập trình di truyền, do đó tăng thêm hiệu quả của chúng.

PHỤ LỤC

Phụ lục 1

CÁC CHỦ ĐỀ CHỌN LỌC

Với một bài toán được hình thức hóa tốt, ta có thể chứng minh được thuật giải di truyền hội tụ được về lời giải tối ưu. Tuy nhiên, các ứng dụng thực tiễn thường khác xa lý thuyết, do:

- Cách biểu diễn nhiễm sắc thể có thể tạo ra không gian tìm kiếm khác với không gian thực sự của bài toán;
- Số bước lặp, khi cài đặt, phải xác định trước, và
- Kích thước quần thể giới hạn.

Những quan sát này hàm ý rằng, trong một số trường hợp, GA không thể tìm được lời giải tối ưu; nguyên nhân chính là do GA hội tụ sớm về các lời giải tối ưu cục bộ. Hội tụ sớm là vấn đề lớn của thuật giải di truyền cũng như các thuật giải tối ưu hóa khác. Nếu hội tụ xảy ra quá nhanh, thì các thông tin đáng giá đã phát triển trong quần thể thường bị mất mát.

Eshelman và Schaffer đã giới thiệu một số chiến lược mới nhằm chống hội tụ sớm; đó là (1) chiến lược ghép đôi, được gọi là *tránh hỗn giao*, (2) sử dụng phép lai đồng dạng, và (3) khám phá những chuỗi trùng lắp trong quần thể.

Tuy nhiên, hầu hết những nghiên cứu về lĩnh vực này đều liên quan đến:

- Quy mô và loại sai số do cơ chế tạo mẫu và
- Bản chất của hàm mục tiêu.

1. CƠ CHẾ TẠO MẪU

Dường như có hai vấn đề quan trọng trong tiến trình tiến hóa của tìm kiếm di truyền: *Tính đa dạng quần thể* và *áp lực chọn lọc*. Những yếu tố này liên quan mạnh mẽ với nhau: khi tăng áp lực chọn lọc thì tính đa dạng của quần thể sẽ giảm, và ngược lại. Nói cách khác, áp lực chọn lọc mạnh "ủng hộ" tính hội tụ sớm của tìm kiếm GA; nhưng nếu áp lực chọn lọc yếu có thể làm cho tìm kiếm thành vô hiệu. Như vậy, việc cân đối giữa hai yếu tố này rất quan trọng; các cơ chế tạo mẫu đều có khuynh hướng đạt đến mục đích này.

Công trình đầu tiên, và có lẽ đáng ghi nhận nhất, thuộc về DeJong, vào năm 1975. Ông đã xem xét các biến thể của một chọn lọc đơn giản đã được trình bày trong các chương trước. Biến thể đầu tiên – *mô hình phát triển ưu tú* – bảo tồn nhiễm sắc thể tốt nhất. Biến thể thứ hai – *mô hình giá trị mong đợi* – giảm độ hỗn loạn của chọn lọc. Thực hiện điều này bằng cách đưa vào một biến đếm cho từng nhiễm sắc thể v, với khởi tạo đầu $f(v)/\overline{f}$, và giảm dần xuống 0.5 hay 1.0 khi nhiễm sắc thể được chọn tái sinh cho lai hoặc đột biến. Khi biến đếm nhiễm sắc thể tụt xuống dưới số 0, thì nó không còn thích hợp để được chọn nữa. Biến thể thứ ba – *mô hình giá trị mong đợi ưu tú* – là kết hợp của hai biến thể trên. Mô hình thứ tư – *mô hình nhân tố tập trung* – một nhiễm sắc thể mới sinh thay cho nhiễm sắc thể "cũ" và một nhiễm sắc thể bị tiêu hủy sẽ được chọn trong số nhiễm sắc thể giống với nhiễm sắc thể mới.

Năm 1981, Brindle đã xem xét thêm một số biến thể khác: *tạo mẫu tất định, tạo mẫu hỗn loạn phần dư không thay thế, đấu tranh hỗn loạn, và tạo mẫu hỗn loạn phần dư có thay thế*. Nghiên cứu này xác nhận ưu thế hơn hẳn của các cải tiến so với chọn lọc đơn giản. Đặc biệt, phương pháp tạo mẫu hỗn loạn phần dư có thay thế cấp phát các mẫu, tùy theo các phần nguyên của giá trị cần có của các biến cố xảy ra trong mỗi nhiễm sắc thể trong quần thể mới và nơi các nhiễm sắc thể cạnh tranh tương ứng với phần thập phân đối với

các vị trí còn lại trong quần thể, đây là phương pháp thành công nhất và được nhiều nhà nghiên cứu xem như chuẩn. Năm 1987, Baker đã giới thiệu một nghiên cứu lý thuyết toàn diện về các cải tiến này bằng một số độ đo được định nghĩa tốt, và cũng đưa ra một phiên bản mới gọi là *tạo mẫu không gian hỗn loạn*. Phương pháp này dùng cách "quay" bánh xe định tỉ lệ trước để thực hiện chọn lọc. Bánh xe này được thiết kế theo chuẩn, được quay với một số khoảng chia đều theo kích thước quần thể, khác với những gì thường thấy ở một bánh xe.

Các phương pháp khác để tạo mẫu quần thể là đánh trọng các nhiễm sắc thể: nhiễm sắc thể được chọn theo thứ hạng của chúng thay vì theo giá trị thực. Các phương pháp này dựa trên nhận xét về lý do hội tụ sớm thường do có sự hiện diện của các *siêu cá thể*, những siêu cá thể này tốt hơn khả năng thích nghi trung bình của quần thể nhiều. Những siêu cá thể này có số con nhiều hơn và (do kích thước quần thể không đổi) sẽ ngăn các cá thể khác sinh sản trong các thế hệ kế tiếp. Trong một số thế hệ, một siêu cá thể có thể loại bỏ chất liệu di truyền tốt và làm cho hội tụ sớm về tối ưu (có thể là cục bộ).

Có nhiều cách xếp hạng. Thí dụ, Baker lấy giá trị do người sử dụng định nghĩa, MAX, như cận trên của số con cần có, và một đường cong tuyến tính băng qua MAX, sao cho vùng bên dưới đường này bằng với kích thước quần thể. Theo cách đó ta có thể dễ dàng quyết định sự khác biệt giữa số con mong đợi với các cá thể "kề cận". Thí dụ, với MAX = 2.0 và *pop-size* = 50, khác biệt giữa số con mong đợi và các cá thể "kề cận" sẽ là 0.04 = 2.0/50.

Một khả năng khác là dùng tham số người sử dụng định nghĩa q và định nghĩa một hàm tuyến tính, thí dụ:

$$prop\ (rank) = q - (rank-1)*r\ ,$$

hay một hàm phi tuyến như:

$$prop\ (rank) = (1-q)^{rank-1}.$$

Cả hai hàm đều trả về xác suất cá thể được một xếp hạng *rank* (*rank =1* nghĩa là cá thể tốt nhất, *rank = pop–size* là cá thể xấu nhất) trong một lần chọn.

Cả hai phương sách này đều cho phép người sử dụng tác động vào áp lực chọn lọc của thuật giải. Trong trường hợp hàm tuyến tính, yêu cầu:

$$\sum_{i=1}^{pop_size} prob(i) = 1$$

có nghĩa là:

$$q = r\ (pop\text{–}size-1)/2 + 1/pop\text{–}size$$

nếu $r = 0$ (kéo theo $q = 1/pop\text{–}size$) thì không có áp lực chọn lọc gì cả: tất cả các cá thể được chọn theo cùng xác suất. Mặt khác, nếu

$$q - (pop\text{–}size-1)*r = 0$$

thì:

$$r = 2\ /\ (n(n-1)),\ và\ q = 2\ /\ n,$$

cho áp lực chọn lọc lớn nhất.

Nói cách khác, nếu một hàm tuyến tính được chọn để tạo xác suất cho cá thể được xếp hạng, một tham số riêng *q*, biến thiên trong khoảng [1/*pop–size*, 2/*pop–size*] có thể kiểm soát áp lực chọn lọc của thuật giải. Thí dụ, nếu *pop–size* = 100, và *q* = 0.015 thì *r* =

q/(*pop–size–1*) = 0.00015151515, và *prop(1)*= 0.015, *prop(2)* = 0.0148484848, ...

prop(100) = 0.00000000000000000051.

Đối với hàm phi tuyến, tham số $q \in (0..1)$ không phụ thuộc kích thước quần thể; giá trị *q* lớn hơn có nghĩa là áp lực chọn lọc của thuật giải mạnh hơn. Thí dụ, nếu *q =0.1* và *pop–size* = 100, thì *prop(1)* =0.100 và *prop(2)* =0.1 * 0.9 =0.090, *prop(3)* =0.1 *0.9 *0.9= 0.081, ..., *prop(100)* =0.000003. Chú ý rằng:

$$\sum_{i=1}^{pop_size} prob(i) = 1 = \sum_{i=1}^{pop\text{-}size} q(1-q)^{i-1} \approx 1$$

(Có thể thay dấu ≈ trong công thức trên bằng dấu =; khi ấy, chỉ cần định nghĩa $prop(i) = c*q(1-q)^{i-1}$ với $c = \dfrac{1}{1-(1-q)^{pop_size}}$)

Những cách tiếp cận như vậy, mặc dù được đưa ra để cải tiến hành vi của thuật giải di truyền trong một số trường hợp, nhưng cũng có những khuyết điểm. Trước hết, chúng buộc người sử dụng phải quyết định lúc nào thì dùng đến các cơ chế này. Thứ hai, chúng bỏ qua các thông tin có liên quan giữa các nhiễm sắc thể khác nhau. Thứ ba, chúng xử sự với tất cả các trường hợp đều như nhau, bất chấp tầm quan trọng của từng bài toán cụ thể. Cuối cùng, các thủ tục chọn theo xếp hạng vi phạm lý thuyết Lược đồ. Tuy nhiên, chúng giúp tránh vấn đề định tỉ lệ (sẽ được thảo luận trong phần kế tiếp), chúng kiểm soát áp lực chọn lọc tốt hơn, và (cặp đôi với việc sinh sản mỗi–lần–một–con) làm cho việc nghiên cứu tập trung nhiều hơn.

Một phương pháp chọn lọc bổ sung, *chọn lọc tranh đua*, kết hợp ý tưởng xếp hạng theo nhiều cách hiệu quả và rất đáng quan tâm. Phương pháp này (trong một lần lặp) sẽ chọn một số *k* cá thể và chọn ra từ tập *k* phần tử này phần tử tốt nhất cho thế hệ kế tiếp.

Tiến trình này được lặp lại *pop-size* lần. Rõ ràng, nếu *k* lớn sẽ làm tăng áp lực chọn lọc; thường *k = 2* (kích thước tranh đua). Ở đây có thể thêm vào hương vị cho bài toán mô phỏng luyện thép bằng chọn lọc Boltzmann có cân nhắc, trong đó hai phần tử *i, j* tranh đua nhau, và kẻ thắng được xác định qua công thức:

$$\frac{1}{1+e^{\frac{f(i)-f(j)}{T}}}$$

với *T* là nhiệt độ và *f(i)* và *f(j)* lần lượt là các trị của hàm mục tiêu của *i* và *j* (công thức dành cho bài toán cực tiểu hóa).

Back và Hoffmeister đã khảo sát về các đặc trưng của thủ tục chọn lọc. Họ chia các thủ tục chọn lọc thành hai loại: *động* và *tĩnh* – chọn lọc tĩnh yêu cầu xác suất chọn lọc là một hằng cho mọi thế hệ (thí dụ, chọn lọc theo hạng), trong khi chọn lọc động thì không cần điều này (nghĩa là, chọn lọc theo tỉ lệ). Một cách phân loại khác là chia các thủ tục thành các loại *tiêu diệt* và *bảo tồn* – chọn lọc bảo tồn yêu cầu có xác suất chọn lọc *không bằng* 0 đối với mỗi cá thể, trong khi chọn lọc tiêu diệt thì không. Các thủ tục chọn lọc tiêu diệt lại được chia làm hai loại *phải* và *trái*: trong chọn lọc tiêu diệt trái, các cá thể tốt nhất bị ngăn không cho sinh sản để tránh hội tụ sớm do các siêu cá thể (chọn lọc phải thì không). Ngoài ra, một số thủ tục chọn lọc *thuần chủng*, nghĩa là cha–mẹ chỉ được phép sinh con trong một thế hệ thôi (nghĩa là, thời gian sống của mỗi cá thể bị giới hạn trong một thế hệ thôi bất kể độ thích nghi của nó). Chúng ta sẽ trở lại vấn đề chọn lọc thuần chủng và tiêu diệt trong phụ lục 2, khi bàn về các chiến lược tiến hóa và so sánh chúng với các thuật giải di truyền. Một số chọn lọc có *tính thế hệ* theo nghĩa là tập các cha–mẹ được giữ cố định cho đến khi tất cả các con của thế hệ tiếp theo được sinh ra hoàn toàn; trong các cách chọn lọc *trên đường tiến* thì con sẽ lập tức thay thế cha–mẹ của nó. Một số chọn lọc là ưu tú theo nghĩa

là một số (hay tất cả) cha–mẹ được phép đồng thời qua chọn lọc cùng với con của chúng.

Trong hầu hết các thử nghiệm được trình bày trong phần này, ta dùng một biến thể mới, gồm hai bước, của thuật giải chọn lọc cơ bản. Nhưng hiệu chỉnh này không chỉ là một cơ chế chọn lọc mới: nó có thể sử dụng bất cứ phương thức tạo mẫu nào và chính nó cũng được thiết kế để giảm những tác động ngoài ý muốn của một số đặc trưng của hàm. Nó rơi vào loại chọn lọc động, bảo tồn, có tính thế hệ và ưu tú.

Cấu trúc của thuật giải di truyền cải tiến (modGA) được trình bày trong hình p1. Cải tiến thuật giải di truyền cổ điển là trong modGA, ta không phải thực hiện bước chọn lọc "tái sinh *P'(t)* từ *P(t-1)*", mà chọn các nhiễm sắc thể *r* một cách độc lập (không cần thiết phải khác biệt) để sinh con và các nhiễm sắc thể (phân biệt) bị loại. Sau khi các bước "chọn–cha–mẹ" và "chọn–loại" của modGA đã được thực hiện, có ba nhóm chuỗi (không nhất thiết rời rạc) trong quần thể:

- *r* chuỗi (không nhất thiết phân biệt) để sinh sản (cha–mẹ),
- chính xác *r* chuỗi bị loại, và
- các chuỗi còn lại gọi là chuỗi trung tính.

Số chuỗi trung tính trong một quần thể (ít nhất là *pop-size – 2r* và nhiều nhất là *pop-size–r)* tuỳ thuộc vào số cha–mẹ đã chọn lọc phân biệt và số các chuỗi gối lên nhau trong loại "cha–mẹ" và loại "loại". Rồi quần thể mới *P(t+1)* được tạo ra, cần có *pop-size – r* chuỗi (tất cả các chuỗi, trừ những chuỗi đã chọn để loại) và *r* con của *r* cha–mẹ.

Thủ tục modGA

bắt đầu

t ← 0

khởi tạo P(t)

lượng giá P(t)

khi (điều kiện – dừng chưa thỏa) **làm**

bắt đầu

t ← t+1

chọn cha–mẹ từ P(t–1)

chọn các cá thể loại từ P(t–1)

tạo P(t): tái sinh cha–mẹ

lượng giá P(t)

hết khi

hết modGA

Hình P.1. Thuật giải modGA

Như đã trình bày, thuật giải có một bước khó giải quyết: làm sao chọn *r* nhiễm sắc thể để loại. Rõ ràng ta muốn thực hiện chọn lọc này theo cách mà các nhiễm sắc thể mạnh hơn ít khả năng bị loại hơn. Muốn thế, ta thay đổi phương thức tạo quần thể mới *P(t+1)* như sau:

bước 1: chọn *r* cha–mẹ từ *P(t)*. Mỗi nhiễm sắc thể đã chọn (hay mỗi bản sao của một số nhiễm sắc thể được chọn) được đánh dấu là có thể áp dụng chính xác cho một toán tử di truyền cố định,

bước 2: chọn (*pop–size – r*) nhiễm sắc thể khác nhau từ *P(t)* và sao chép chúng vào *P(t+1)*

bước 3: cho *r* nhiễm sắc thể cha–mẹ, để có thể sinh được chính xác *r* con

bước 4: chèn *r* con mới này vào quần thể *P(t+1)*

Những chọn lọc trên (ở bước 1 và 2) được thực hiện tùy thuộc độ thích nghi của các nhiễm sắc thể (phương pháp tạo mẫu không gian hỗn loạn). Có một số khác biệt quan trọng giữa các chương trình chọn lọc khác nhau đã được bàn đến trước đây và chương trình vừa trình bày ở trên. Trước tiên, cả cha–mẹ và con đều có nhiều cơ may để hiện hữu trong thế hệ mới: một cá thể trên trung bình sẽ có nhiều cơ hội để được chọn làm cha–mẹ (bước 1) và, đồng thời, được chọn trong quần thể mới của (*pop–size–r*) phần tử (bước 2). Nếu như thế, một hoặc nhiều con của nó sẽ nhận một số trong *r* vị trí còn lại. Thứ hai, ta áp dụng các toán tử di truyền trên toàn bộ các cá thể tương phản với các bit cá thể (đột biến cổ điển). Điều này có thể tạo ra cách xử lý đồng nhất tất cả các toán tử được dùng trong chương trình tiến hóa. Như vậy, nếu ba toán tử được dùng (đột biến, lai tạo, đảo), một số cha–mẹ sẽ qua đột biến, một số khác qua lai tạo và số còn lại bị đảo.

Cách tiếp cận modGA có một số đặc tính lý thuyết như thuật giải di truyền cổ điển. Ta có thể viết lại phương trình tăng trưởng (3.3) trong chương 3 thành:

$$\xi(S, t+1) \geq \xi(S, t) * p_s(S) * p_g(S) \qquad (1)$$

trong đó $p_s(S)$ biểu diễn xác suất sinh tồn của lược đồ S và $p_g(S)$ biểu diễn tăng trưởng của lược đồ S. Tăng trưởng của lược đồ S xảy ra trong giai đoạn chọn lọc (giai đoạn tăng trưởng) khi nhiều bản sao của các lược đồ trên trung bình được chép vào quần thể mới. Xác suất tăng trưởng $p_s(S)$ của lược đồ S, $p_g(S) = eval(S,t)/\overline{F(t)}$, *và* $p_s(S)>1$ đối với các lược đồ tốt trên trung bình. Rồi các nhiễm sắc thể được chọn phải sống sót qua lai tạo và đột biến các toán tử di truyền (giai đoạn co rút). Như đã nói trong chương 3, xác suất sinh tồn $p_s(S)$ của lược đồ S, là:

$$p_s(S) = 1 - p_c \frac{\delta(S)}{m-1} - p_m.o(S) < 1$$

Phương trình (1) hàm ý là lược đồ ngắn, bậc thấp, $p_g(S) * p_s(S)>1$; do đó mà những lược đồ như thế nhận số lần thử tăng theo lũy thừa trong những thế hệ kế tiếp. Điều tương tự cũng đúng cho phiên bản modGA. Số nhiễm sắc thể cần có của lược đồ S trong thuật giải modGA cũng là kết quả của số nhiễm sắc thể trong quần thể cũ $\xi(S, t)$, xác suất sinh tồn ($p_s(S)$ <1) và xác suất tăng trưởng $P_g(s)$ – điểm khác biệt duy nhất là sự thông dịch các thời kỳ tăng trưởng và co rút và bậc tương đối của chúng. Trong phiên bản modGA, thời kỳ co rút là: $n - r$ nhiễm sắc thể được chọn cho quần thể mới. Xác suất sinh tồn được định nghĩa là một phần các nhiễm sắc thể của lược đồ S không bị chọn loại. Thời kỳ tăng trưởng xảy ra tiếp đó và được biểu hiện trong việc xuất hiện của r con mới. Xác suất tăng trưởng $p_g(S)$ của lược đồ S là xác suất tăng của lược đồ này, do các con mới sinh ra từ r cha–mẹ. Lần nữa, đối với các lược đồ

ngắn, bậc thấp, $p_g(S)*p_s(S)>1$ vẫn đúng và những lược đồ đó nhận các lần thử tăng theo lũy thừa trong các thế hệ kế tiếp.

Một trong những ý tưởng về thuật giải modGA là việc sử dụng các tài nguyên lưu trữ sẵn tốt hơn: kích thước quần thể. Thuật giải mới tránh được việc để lại những bản sao y hệt của cùng các nhiễm sắc thể trong các quần thể mới (điều này cũng có thể xảy ra ngẫu nhiên, nhưng thường rất hiếm). Mặt khác, thuật giải cổ điển rất dễ tạo nhiều bản sao như thế. Hơn nữa, nhiều sự cố như thế về các siêu cá thể tạo xác suất cho một phản ứng dây chuyền: có cơ hội cho một số lượng lớn hơn các bản sao y như thế trong quần thể kế tiếp v.v... Cách này khiến kích thước quần thể bị giới hạn có thể thực sự chỉ biểu diễn một số giảm của các nhiễm sắc thể đồng nhất. Sử dụng không gian nhỏ sẽ làm giảm hiệu quả của thuật giải. Chú ý rằng cơ sở lý thuyết của thuật giải di truyền giả định kích thước quần thể vô hạn. Trong thuật giải modGA, ta có thể có một số thành viên của gia đình nhiễm sắc thể, nhưng tất cả các thành viên như thế đều khác nhau (dùng từ ***gia đình*** là ta muốn nói đến những con của cùng cha mẹ).

Như một thí dụ về một nhiễm sắc thể với một giá trị mong đợi trong $P(t+1)$ bằng $p = 3$. Cũng giả định là thuật giải di truyền cổ điển có xác suất lai tạo và đột biến $p_c = 0.3$ và $p_m = 0.003$. Sau bước chọn lọc, sẽ có chính xác $p = 3$ bản sao nhiễm sắc thể này trong quần thể $P(t+1)$ trước khi sinh. Sau khi sinh, cho chiều dài nhiễm sắc thể bằng 20, số bản sao y hệt của nhiễm sắc thể này còn lại trong $P(t+1)$ sẽ là $p*(1-p_c - p_m * m) = 1.92$. Do đó, có thể an tâm khi nói rằng quần thể kế tiếp sẽ có hai bản sao y của nhiễm sắc thể đó, giảm đi số nhiễm sắc thể khác.

Các cải tiến trong modGA dựa trên ý tưởng về mô hình nhân tố tụ tập, trong mô hình này nhiễm sắc thể mới phát sinh sẽ thay thế một nhiễm sắc thể cũ nào đó. Nhưng khác biệt là ở chỗ trong mô hình nhân tố tụ tập nhiễm sắc thể bị loại được chọn từ số nhiễm sắc

thể giống nhiễm sắc thể mới, còn trong modGA, các nhiễm sắc thể bị loại là những nhiễm sắc thể có độ thích nghi thấp hơn.

Đối với r nhỏ, modGA thuộc về lớp các Thuật giải Tiến Hóa Trạng Thái Ổn Định (SSGA: Steady State GAs); khác biệt chính giữa GA va SSGA là trong SSGA chỉ một số ít thành viên của quần thể bị thay đổi (trong từng thế hệ). Cũng có vài tương đồng giữa modGA và các hệ phân lớp: thành phần di truyền của hệ phân lớp làm thay đổi quần thể càng ít càng tốt. Trong modGA, ta có thể điều chỉnh một thay đổi như thế nhờ tham số r, nó xác định số nhiễm sắc thể cần sinh ra và số nhiễm sắc thể phải chết. Trong modGA, $(pop_size - r)$ nhiễm sắc thể được đặt vào quần thể mới mà không biến đổi. Đặc biệt, khi $r = 1$, chỉ một nhiễm sắc thể được thay thế trong mỗi thế hệ. Gần đây, Miihlenbein đề xuất thuật giải tái sinh GA (BGA: breeder GA), trong đó r cá thể tốt nhất được chọn và ghép đôi ngẫu nhiên cho đến khi số con tương đương với kích thước quần thể. Thế hệ con thay thế quần thể cha-mẹ và cá thể tốt nhất tìm thấy từ trước đến giờ vẫn còn lại trong quần thể.

2. Các đặc trưng của hàm

Thuật giải modGA cung cấp một cơ chế mới tạo một quần thể mới từ quần thể cũ. Tuy nhiên dường như một số độ đo bổ sung có lẽ liên quan đến đặc trưng hàm đang cần tối ưu. Qua nhiều năm, ta đã thấy được ba hướng cơ bản. Một trong các hướng vay mượn kỹ thuật mô phỏng luyện thép là thay đổi độ hỗn loạn của hệ thống. (tốc độ hội tụ quần thể được kiểm soát bằng các toán tử nhiệt động học, các toán tử này sử dụng tham số nhiệt độ toàn cục).

Một hướng khác là chỉ định tái sinh tương ứng với thứ hạng thay vì theo giá trị thực (đã thảo luận trong đoạn trước), do việc xếp hạng một cách tự động dẫn đến việc định tỉ lệ đồng nhất qua quần thể.

Hướng cuối cùng tập trung vào việc thử cố định chính hàm cần tối ưu bằng cách đưa vào một cơ chế định tỉ lệ. Theo Goldberg, ta chia các cơ chế này thành ba loại:

1. *Định tỉ lệ tuyến tính*. Theo phương pháp này, độ thích nghi các nhiễm sắc thể hiện có được định theo công thức:

$$f'_i = a^* f_i + b,$$

các tham số a, b được chọn sao cho độ thích nghi trung bình được ánh xạ vào chính nó và tăng độ thích nghi tốt nhất bằng cách nhân với độ thích nghi trung bình. Cơ chế này, dù rất mạnh, có thể tạo ra các giá trị âm cần phải xử lý riêng. Ngoài ra các tham số a, b thường gắn với đời sống quần thể và không phụ thuộc bài toán.

2. *Phép cắt Sigma*. Phương pháp được thiết kế là một cải tiến về định tỉ lệ tuyến tính vừa để xử lý các giá trị âm, vừa để kết hợp thông tin phụ thuộc bài toán vào chính ánh xạ. Ở đây, độ thích nghi mới được tính theo:

$$f'_i = f_i + (\overline{f} - c^* \sigma),$$

trong đó c là số nguyên nhỏ (thường là một số trong khoảng 1 đến 5) còn σ là độ lệch chuẩn của quần thể; với giá trị âm thì f' được thiết lập bằng 0.

3. *Định tỉ lệ luật lũy thừa*. Trong phương pháp này, thích nghi lúc khởi tạo được coi như năng lực đặc biệt:

$$f'_i = f_i^k,$$

với một số k gần bằng 1. Tham số k định tỉ lệ hàm f ; tuy nhiên, một số nhà nghiên cứu cho rằng nên chọn k độc lập với bài toán. Cũng trong các nghiên cứu đó, các tác giả đã cho $k = 1.005$.

Vấn đề đáng lưu tâm nhất, kết hợp với đặc trưng của hàm đang được quan tâm có nhiều khác biệt trong độ thích nghi tương đối. Để minh họa, xét hai hàm: $f_1(x)$ và $f_2(x) + c$ *(hằng số)*, vì cả hai cơ bản giống nhau (nghĩa là, chúng có cùng một lời giải tối ưu), ta có thể cho rằng cả hai có thể được tối ưu hóa với cùng độ phức tạp. Nhưng, nếu $c >> f_1(x)$, thì hàm $f_2(x)$ sẽ hội tụ chậm hơn hàm $f_1(x)$ nhiều. Thật ra, ở trường hợp thái quá, hàm thứ hai sẽ được tối ưu hóa bằng một tiến trình tìm kiếm hoàn toàn ngẫu nhiên; cách ứng phó như vậy có lẽ chấp nhận được trong đời sống ban đầu của quần thể, nhưng về sau nó có thể gây hại. Ngược lại, $f_1(x)$ có thể hội tụ quá nhanh đẩy thuật giải về tối ưu cục bộ.

Ngoài ra, do kích thước của quần thể cố định, cách ứng phó của một GA có thể không giống nhau trong mỗi lần chạy – đó là do các lỗi tạo mẫu hữu hạn. Xét hàm $f_3(x)$ với mẫu $x^t_i \in P(t)$ gần một số tối ưu cục bộ và $f_1(x^t_i)$ lớn hơn thích nghi trung bình $\bar{f}(x^t)$ nhiều (nghĩa là x^t_i là siêu cá thể). Hơn nữa, giả định rằng không có x^t_j gần cực đại toàn cục đã tìm được. Có thể đây là trường hợp của hàm không–trơn cao độ. Trong trường hợp như vậy, sẽ có sự hội tụ nhanh về tối ưu cục bộ. Do vậy, quần thể $P(t+1)$ trở nên quá bão hòa với các phần tử gần lời giải đó, làm giảm cơ hội khai thác toàn cục cần cho việc tìm kiếm các tối ưu khác. Trong khi cách ứng phó như vậy là chấp nhận được ở các giai đoạn tiến hóa sau này, và thậm chí đáng ao ước ở những giai đoạn cuối, thì ở những giai đoạn đầu nó lại hoàn toàn gây phiền phức. Hơn nữa, bình thường các quần thể về sau (trong các giai đoạn cuối của thuật giải) bão hòa với những nhiễm sắc thể có cùng độ thích nghi, do tất cả các nhiễm sắc thể này có liên quan rất mật thiết (bởi các tiến trình giao phối). Vì thế, bằng các kỹ thuật chọn lọc truyền thống, việc tạo mẫu thực sự trở nên ngẫu nhiên. Cách ứng phó đó hoàn toàn trái với cách ứng phó đáng mong đợi nhất, mà theo đó, tác động của thích nghi của các nhiễm sắc thể tương đối giảm dần trong tiến trình chọn lọc ở các giai đoạn khởi đầu của đời sống quần thể và tăng ở những giai đoạn cuối.

Một trong những hệ thống nổi tiếng nhất, GENESIS 1.2ucsd, dùng 2 tham số để điều khiển việc tìm kiếm có liên quan đến đặc trưng của hàm được tối ưu hóa: cửa sổ định tỉ lệ và hạng tử của phép cắt sigma. Hệ thống cực tiểu hóa hàm, trong những trường hợp như vậy, thường thì hàm lượng giá *eval* trả về:

$$eval(x) = F - f(x),$$

trong đó, F là hằng số thỏa $F > f(x)$, với mọi x. Như đã nói ở trên, việc chọn F xấu sẽ có tác động không tốt cho việc tìm kiếm, hơn nữa F có thể không thích hợp. Cửa sổ định tỉ lệ W của GENESIS 1.2ucsd cho phép người sử dụng điều khiển kỳ cập nhật cho hằng số F: nếu $W > 0$ hệ thống đặt F là cực trị của $f(x)$ đã xuất hiện trong các thế hệ W cuối cùng. Giá trị của $W = 0$ biểu diễn cửa sổ vô hạn, nghĩa là $F = max\ \{f(x)\}$ qua tất cả các lần tiến hóa. Nếu $W < 0$, người sử dụng có thể dùng một phương pháp khác đã được nói đến từ trước: phép cắt sigma.

Cũng cần khảo sát tầm quan trọng của điều kiện dừng đơn giản nhất có thể kiểm soát số thế hệ hiện tại; tìm kiếm sẽ kết thúc nếu tổng số thế hệ đã vượt quá một hằng số chọ trước. Theo hình 0.1 (phần dẫn nhập), điều kiện kết thúc đó được biểu diễn là "$t \geq T$" đối với vài hằng số T. Trong một số phiên bản về chương trình tiến hóa, không phải mọi cá thể đều cần tiến hóa lại: vài cá thể trong đó vượt từ thế hệ này qua thế hệ kế tiếp mà chẳng thay đổi gì cả. Trong những trường hợp như thế, sẽ rất ý nghĩa (để so sánh với một số thuật giải cổ điển khác) nếu ta đếm số lần lượng giá hàm (thường thì con số này tỉ lệ với số thế hệ) và kết thúc tìm kiếm khi số lần lượng giá hàm vượt quá một hằng đã định trước.

Nhưng những điều kiện kết thúc ở trên giả thiết như người sử dụng đã biết đặc trưng của hàm, có ảnh hưởng đến chiều dài của tìm kiếm. Trong nhiều thí dụ, rất khó xác định tổng số thế hệ (hay lượng giá hàm) phải là bao nhiêu. Có lẽ sẽ tốt hơn nhiều nếu thuật

giải chấm dứt quá trình tìm kiếm khi cơ hội cho một cải thiện quan trọng còn tương đối mong manh.

Có hai loại điều kiện dừng cơ bản, các điều kiện này dùng các đặc trưng tìm kiếm để quyết định ngừng quá trình tìm kiếm. Một loại – dựa trên cấu trúc nhiễm sắc thể (kiểu gen); loại kia – dựa trên ý nghĩa của một nhiễm sắc thể đặc biệt. Các điều kiện dừng trong loại thứ nhất đo sự hội tụ của quần thể bằng cách kiểm soát số alen được hội tụ, ở đây alen được coi như hội tụ nếu một số phần trăm quần thể đã định trước có cùng (hoặc tương đương – đối với các biểu diễn không nhị phân) giá trị trong alen này. Nếu số alen hội tụ vượt quá số phần trăm nào đó của tổng số alen, việc tìm kiếm sẽ kết thúc. Các điều kiện kết thúc trong loại thứ hai đo tiến bộ của thuật giải trong một số thế hệ đã định trước: nếu tiến bộ này nhỏ hơn một hằng số epsilon nào đó (được cho như một tham số của phương pháp), kết thúc tìm kiếm.

3. THUẬT GIẢI DI TRUYỀN ÁNH XẠ CO

Sự hội tụ của thuật giải di truyền là một trong những vấn đề lý thuyết nhiều thách thức nhất trong lãnh vực máy tính tiến hóa. Nhiều nhà khảo cứu khai thác vấn đề này từ nhiều góc cạnh khác nhau. Golberg và Segrest sử dụng xích Markov hữu hạn phân tích thuật giải di truyền (quần thể hữu hạn, chỉ sinh sản và đột biến). David và Principe khám phá một khả năng ngoại suy của nền tảng lý thuyết đang có về thuật giải mô phỏng luyện thép trên mô hình thuật giải di truyền và xích Markov. Eiben, Aarts và Van Hee đã đề nghị thuật giải di truyền trừu tượng thống nhất thuật giải di truyền và mô phỏng luyện thép; người ta đã thảo luận về phân tích xích Markov trên thuật giải di truyền trừu tượng như vậy và những điều kiện hàm ý rằng quá trình tiến hóa tìm ra tối ưu với xác suất bằng 1 đã cho trước. Kingdon tìm ra các điểm khởi đầu, hội tụ và lớp các bài toán mà các thuật giải di truyền khó giải được. Ý niệm về các lược đồ đối kháng được tổng quát hóa và xác suất hội tụ của những

lược đồ được cung cấp. Nhiều nhà nghiên cứu cũng xét nhiều loại định nghĩa khác nhau về các bài toán lừa. Gần đây Rudov chứng minh rằng thuật giải di truyền cổ điển không bao giờ hội tụ về tối ưu toàn cục, mà chính các phiên bản được hiệu chỉnh đã bảo tồn lời giải tốt nhất trong quần thể (như mô hình ưu tú).

Có thể tiếp cận định lý điểm bất động Banăc để chứng minh sự hội tụ của thuật giải di truyền. Tiếp cận này cung cấp một cách giải thích trực giác về tính hội tụ của các GA (không có mô hình ưu tú); yêu cầu duy nhất là phải có cải thiện trong quần thể kế tiếp (không nhất thiết là cải thiện của cá thể tốt nhất). Định lý điểm bất động Banăc sử dụng ánh xạ co trong không gian mêtric. Định lý này phát biểu rằng, bất cứ ánh xạ co f nào cũng có một điểm bất động duy nhất, nghĩa là, một phần tử x sao cho $f(x) = x$. Kỹ thuật điểm bất động thường được coi là công cụ mạnh để định nghĩa ngữ nghĩa máy tính. Thí dụ, các ngữ nghĩa biểu thị của một chương trình hoặc máy tính thường được cho là điểm bất động tối thiểu của ánh xạ liên tục được định nghĩa trên một dàn thích hợp. Nhưng không giống các ngữ nghĩa biểu thị truyền thống, ta thấy rằng đây là những không gian mêtric cung cấp một phương cách tự nhiên và rất đơn giản để biểu diễn ngữ nghĩa của thuật giải di truyền. Thuật giải di truyền có thể được định nghĩa là sự biến dạng giữa các quần thể. Bây giờ, giả sử ta có thể tìm được những không gian mêtric như vậy mà trong các không gian này biến dạng đó là co. Trong trường hợp đó, ngữ nghĩa của thuật giải di truyền được xem là các điểm bất động của biến đổi cấp trên. Do bất cứ biến dạng nào như vậy cũng cố duy nhất một điểm bất động, nên sự hội tụ của thuật giải di truyền chỉ là một hệ quả đơn giản.

Theo trực giác, một không gian mêtric là một cặp thứ tự của một tập và một hàm cho phép ta đo khoảng cách giữa các cặp phần tử của tập. Một ánh xạ f được định nghĩa trên các phần tử của một tập như thế là ánh xạ co nếu khoảng cách giữa $f(x)$ và $f(y)$ nhỏ hơn khoảng cách giữa x và y.

Bây giờ, ta định nghĩa một cách hình thức hơn. Gọi R là tập các số thực. Tập S và ánh xạ $\delta : S \times S \rightarrow R$ là *không gian mêtric* nếu với mọi $x,y \in S$:

- $\delta(x,y) \geq 0$ và $\delta(x,y) = 0$ nếu $x = y$
- $\delta(x,y) = \delta(y,x)$
- $\delta(x,y) + \delta(y,z) \geq \delta(x,z)$

Ánh xạ δ được gọi là khoảng cách. Ta thường biểu thị không gian mêtric bằng cặp $<S,\delta>$.

Cho $<S,\delta>$ là không gian mêtric và $f: S \rightarrow S$ là một ánh xạ. Ta nói f là ánh xạ co nếu và chỉ nếu có một hằng số $\varepsilon \in [0,1)$ sao cho với mọi $x,y \in S$:

$$(f(x), f(y)) \leq \varepsilon * \delta(x,y)$$

Để hình thức hóa định lý Banăc, ta phải định nghĩa tính đủ của không gian mêtric. Dãy $p_0, p_1, \ldots$ các phần tử thuộc không gian mêtric $<S,\delta>$ là dãy Cauchy nếu và chỉ nếu với mọi $\varepsilon > 0$, tồn tại k sao cho với mọi $m, n > k$, $\delta(p_m, p_n) < \varepsilon$. Không gian mêtric là đủ nếu mọi dãy Cauchy $p_0, p_1, \ldots$ đều có giới hạn $p = \lim_{n \to \infty} p_n$.

Và đây là định lý Banăc. Việc chứng minh định lý có thể tìm thấy trong hầu hết tài liệu về tôpô.

Định lý. Cho $<S, \delta>$ là không gian mêtric đủ và $f: S \rightarrow S$ là ánh xạ co. Thì f có điểm cố định duy nhất $x \in S$ sao cho với mọi $x_0 \in S$:

$$x = \lim_{i \to \infty} f^i(x_0)$$

trong đó $f^0(x_0) = x_0$ và $f^{i+1}(x_0) = f(f^i(x_0))$

Định lý Banăc có thể sử dụng để chứng minh tính hội tụ của thuật giải di truyền. Thực vậy, nếu ta xây dựng không gian mêtric S theo cách mỗi phần tử của không gian là các quần thể, thì bất cứ ánh xạ co f nào cũng có điểm bất động duy nhất, mà theo định lý Banăc, điểm này đạt được bằng việc lặp f được áp dụng vào một quần thể khởi đầu được chọn ngẫu nhiên $P(0)$. Như vậy, ta tìm được không gian mêtric thích hợp mà trong đó thuật giải di truyền co, rồi ta có thể chứng minh sự hội tụ của các thuật giải di truyền đó về điểm bất động, độc lập với việc chọn khởi tạo quần thể ban đầu. Việc chứng minh rất đơn giản với một chút biến thể của thuật giải di truyền, gọi là Thuật giải Di Truyền Ánh Xạ Co (CM-GA).

Không mất tính tổng quát, giả sử ta đang xét bài toán cực đại hóa, nghĩa là các bài toán mà lời giải $\overline{x_i}$ tốt hơn lời giải $\overline{x_j}$ nếu và chỉ nếu $eval(\overline{x_i}) > eval(\overline{x_j})$.

Giả sử kích thước của quần thể *pop-size* = n không đổi; mỗi quần thể gồm có n cá thể, nghĩa là $P = \{\overline{x}_1, \ldots, \overline{x}_n\}$. Hơn nữa, ta xét hàm lượng giá *Eval* của quần thể P; thí dụ ta có thể giả định:

$$Eval(P) = \frac{1}{n} \sum_{\overline{x_i} \in P} eval(\overline{x_i})$$

eval trả về độ 'thích nghi' của cá thể x_i trong quần thể P. Tập S là tập gồm tất cả các quần thể có thể có, P, nghĩa là mọi vectơ $\{\overline{x}_1, \ldots, \overline{x}_n\} \in S$.

Tiếp đến, ta sẽ định nghĩa ánh xạ (khoảng cách) $\delta : S \times S \rightarrow R$, và ánh xạ co $f: S \rightarrow S$ trong không gian mêtric $< S, \delta >$. Khoảng cách δ trong không gian mêtric S của các quần thể có thể định nghĩa là:

$$\delta(P_1, P_2) = \begin{cases} 0, & (P_1 = P_2) \\ |1 + M - Eval(P_1)| + |1 + M - Eval(P_2)|, & (P_1 \neq P_2) \end{cases}$$

trong đó, M là chặn hạn trên của hàm *eval* trong miền đang xét, nghĩa là, $eval(\overline{x}) \leq M$ với mọi cá thể $\overline{x}$ (do đó $eval(p) \leq M$ với mọi quần thể P có thể có). Thật ra:

- $\delta(P_1, P_2) \geq 0$ với mọi quần thể P1 và P2; hơn nữa δ(P1, P2) = 0 nếu và chỉ nếu $P1=P2$
- $\delta(P_1,P_2) = \delta(P_2,P_1)$, và
- $\delta(P_1,P_2) + \delta(P_2,P_3) = |1+ M - Eval(P_1)| + |1+M - Eval(P_2)| + |1+M - Eval(P_2)| + |1+M - Eval(P_3)| \geq |1+M - Eval(P_1)| + |1+M - Eval(P_3)| = \delta(P_1,P_3)$,

do đó, < S, δ > là không gian mêtric.

Hơn nữa, không gian mêtric < S, δ > là không gian đủ. Do đối với mọi dãy Cauchy P_1, P_2... các quần thể, luôn tồn tại k sao cho với mọi $n > k$, $P_n = P_k$. Điều này có nghĩa là tất cả các dãy Cauchy P_i có một giới hạn khi $i \to \infty$.

Bây giờ ta bàn về ánh xạ co, $f: S \to S$. Đó đơn giản chỉ là một lần lặp của thuật giải di truyền (xem hình 2) đã cho miễn là có cải thiện (theo ý nghĩa của hàm *Eval*) từ quần thể P(t) đến quần thể P(t+1). Trong trường hợp đó, $f(P(t)) = P(t+1)$. Nói các khác, lần lặp thứ t của thuật giải di truyền sẽ là một toán tử ánh xạ co f nếu và chỉ nếu $Eval(P(t)) < Eval(P(t+1))$. Nếu không có cải thiện, ta không đếm lần lặp đó, nghĩa là, ta chạy tiến trình chọn lọc và tái kết hợp lại một lần nữa.

Cấu trúc của thuật giải di truyền biến thể như thế (Thuật giải Di Truyền Ánh Xạ Co - CM-GA) được minh họa trong hình 2. Bước lặp được hiệu chỉnh như thế của CM-GA thực sự thoả mãn yêu cầu

của ánh xạ co. Hiển nhiên là nếu lần lặp $f: P(t) \to P(t+1)$ cải thiện một quần thể theo ý nghĩa của hàm *Eval*, nghĩa là, nếu:

$$Eval(P_1(t)) < Eval(f(P_1(t))) = Eval(P_1(t+1)), \text{ và}$$

$$Eval(P_2(t)) < Eval(f(P_2(t))) = Eval(P_2(t+1))$$

thì

$\delta(f(p1(t)), f(p2(t)) = |1 + M - Eval(f(P1(t)))| + |1 + M - Eval(f(P2(t)))| < |1 + M - Eval(P1(t))| + |1 + M - Eval(P2(t))| = \delta(P1(t), P2(t))$

Thủ tục CM-GA

bắt đầu

t = 0

khởi tạo P(t)

lượng giá P(t)

khi (điều kiện dừng chưa thỏa) làm

bắt đầu ánh xạ co $f: P(t) \to P(t+1)$

t = t+1

chọn P(t) từ P(t-1)

tái kết hợp P(t)

lượng giá P(t)

nếu $Eval(P(t\text{-}1)) \geq Eval(P(t))$

thì t = t-1

hết lặp

Kết thúc

Hình P.2. Thuật giải di truyền ánh xạ co

Tóm lại, CM-GA thỏa mãn các giả thuyết về điểm bất động Banăc: không gian của các quần thể $< S, \delta >$ là không gian mêtric hoàn chỉnh và lần lặp f : P(t) → P(t+1) (cải thiện quần thể theo ý nghĩa của hàm lượng giá *Eval*) là ánh xạ co. Do đó:

$$P^* = \lim_{i \to \infty} f^i(P((0)))$$

nghĩa là, thuật giải CM-GA hội tụ về quần thể P^*, là điểm bất động duy nhất trong không gian của tất cả quần thể.

Rõ ràng, P^* biểu diễn quần thể đạt được tối ưu toàn cục. Chú ý rằng, hàm *Eval* đã được định nghĩa là:

$$Eval(P) = \frac{1}{n} \sum_{\bar{x}_i \in P} eval(\bar{x}_i)$$

nghĩa là, điểm bất động P^* đạt được khi tất cả các cá thể trong quần thể này có cùng giá trị (cực đại toàn cục). Hơn nữa, P* không phụ thuộc quần thể khởi tạo P(0).

Ở đây xuất hiện một vấn đề thú vị khi hàm lượng giá *Eval* có nhiều cực trị. Trong trường hợp đó, thuật giải di truyền ánh xạ co như trên không thực sự là một ánh xạ co, cũng như đối với các quần thể tối ưu *P(1), P(2)*.

$$\delta(f(P1), f(P2)) = \delta(P1, P2).$$

Mặt khác, có thể thấy rõ trong trường hợp này là CM-GA hội tụ về một trong các quần thể tối ưu có thể có. Điều đó, nghĩa là mỗi lần chạy của thuật giải lại hội tụ về một quần thể tối ưu.

Thoạt đầu, kết quả có vẻ bất ngờ: trong trường hợp thuật giải di truyền ánh xạ co, việc chọn quần thể khởi đầu có thể chỉ ảnh hưởng

đến tốc độ hội tụ. Thuật giải di truyền ánh xạ co được đề nghị (trên cơ sở định lý Banăc) sẽ luôn luôn hội tụ về tối ưu toàn cục (với thời gian vô hạn). Nhưng có thể là (ở một giai đoạn nào đó của thuật giải) không có quần thể mới nào được chấp nhận trong một thời gian dài và thuật giải bị quẩn trong việc tìm ra quần thể mới P(t). Nói cách khác, các toán tử đột biến và lai tạo áp dụng cho một quần thể cụ thể không thể sinh ra quần thể "tốt hơn" và thuật giải quẩn khi cố thực hiện bước hội tụ tiếp theo. Việc chọn khoảng cách δ giữa các quần thể và hàm lượng giá *eval* được thực hiện chỉ để làm đơn giản mọi việc càng nhiều càng tốt. Mặt khác, những chọn lựa như thế có thể ảnh hưởng đến tốc độ hội tụ và có vẻ như phụ thuộc ứng dụng.

4. Thuật giải di truyền với kích thước quần thể thay đổi

Kích thước quần thể là một trong những lựa chọn quan trọng nhất mà tất cả người sử dụng thuật giải di truyền phải đương đầu và có thể tuỳ thuộc ứng dụng. Nếu kích thước quần thể quá nhỏ, thuật giải di truyền hội tụ quá mau; còn nếu quá lớn, sẽ hao phí tài nguyên máy tính: thời gian chờ của một cải thiện có thể rất dài. Như đã nói có hai vấn đề quan trọng trong quá trình tiến hóa của thuật giải: tính đa dạng quần thể và áp lực chọn lọc. Hiển nhiên, cả hai yếu tố này đều bị ảnh hưởng bởi kích thước quần thể.

Trong phần này ta bàn về Thuật giải Di Truyền Với Kích Thước Quần Thể Thay Đổi (GAVaPS). Thuật giải này không dùng một biến thể nào của cơ chế chọn lọc đã bàn trước đây (phần 1), mà lại giới thiệu khái niệm về "tuổi" của một nhiễm sắc thể, tương đương với số thế hệ mà nhiễm sắc thể còn "sống". Như vậy, khái niệm tuổi của nhiễm sắc thể thay cho khái niệm chọn lọc và, do nó phụ thuộc vào độ thích nghi của cá thể, nên ảnh hưởng đến kích thước quần thể ở mỗi giai đoạn của tiến trình. Dường như bước tiếp cận đó cũng "tự nhiên" hơn các cơ chế chọn lọc đã nói trước đây nhiều: cuối cùng,

tiến trình định tuổi được biết đến trong tất cả các môi trường tự nhiên.

Hình như phương pháp sử dụng kích thước quần thể thay đổi tương tự với một số chiến lược tiến hóa (phụ lục 2) và các phương pháp khác mà con cái đấu tranh với cha–mẹ để sinh tồn. Nhưng một khác biệt quan trọng là, tất cả các phương pháp khác kích thước quần thể cố định, trong khi kích thước quần thể trong GAVaPS lại biến đổi từng thời kỳ.

Động cơ khác của chương trình này dựa trên nhận xét sau đây: một số nhà nghiên cứu khảo sát khả năng đưa vào những xác suất bổ sung của các toán tử di truyền cho thuật giải di truyền; những kỹ thuật khác như chiến lược tiến hóa cũng kết hợp với xác suất bổ sung những toán tử của nó thời gian trước đây. Có lý khi cho rằng ở những giai đoạn khác nhau của tiến trình tiến hóa các toán tử khác nhau có tầm quan trọng khác nhau và hệ thống phải được phép tự điều chỉnh các tần số và phạm vi. Điều tương tự cũng đúng đối với kích thước quần thể: ở những giai đoạn khác nhau của tiến trình tiến hóa, những kích thước quần thể khác nhau đều có thể 'tối ưu', như vậy việc thử nghiệm với một số luật heuristic để điều chỉnh kích thước quần thể cho đúng với giai đoạn tìm kiếm hiện hành cũng quan trọng.

Thuật giải GAVaPS, vào thời điểm *t* xử lý quần thể *P(t)* các nhiễm sắc thể. Trong bước 'tái kết hợp P(t)', một quần thể phụ trợ mới được tạo (đây là quần thể con). Kích thước của quần thể phụ trợ tỉ lệ với kích thước quần thể gốc; quần thể phụ trợ chứa *AuxPopSize* (t) = [*pop-size* (t)*ρ] nhiễm sắc thể (ta xem tham số ρ là tốc độ sinh sản). Mỗi nhiễm sắc thể trong quần thể có thể được chọn lọc để tái sinh (nghĩa là, đặt các con vào quần thể phụ trợ) với các xác suất bằng nhau, độc lập với giá trị thích nghi của nó. Con được tạo bằng cách áp dụng các toán tử di truyền (lai và đột biến) vào các nhiễm sắc thể được chọn. Do việc chọn các nhiễm sắc thể không phụ thuộc

vào giá trị thích nghi của chúng, nghĩa là không có bước chọn lọc như thế, ta giới thiệu khái niệm về tuổi của nhiễm sắc thể và tham số *thời gian sống* của nó.

Cấu trúc GAVaPS được trình bày trong hình 3

Tham số thời gian sống được gán một lần cho các nhiễm sắc thể trong bước lượng giá (sau khi khởi tạo các nhiễm sắc thể hoặc sau bước kết hợp các thành viên của quần thể phụ trợ) và giữ nguyên (đối với một nhiễm sắc thể đã cho) qua suốt quá trình tiến hóa, nghĩa là từ lúc sinh cho đến lúc chết của nhiễm sắc thể. Điều này có nghĩa là, đối với nhiễm sắc thể 'già' thì các giá trị thời gian sống của nó *không* được tính lại. Nhiễm sắc thể chết khi tuổi của nó, nghĩa là số thế hệ mà nhiễm sắc thể tồn tại (khởi tạo là 0), vượt quá giá trị thời gian sống của nó. Nói cách khác, thời gian sống của một nhiễm sắc thể quyết định số thế hệ GAVaPS mà trong thời gian đó nhiễm sắc thể được lưu trong quần thể: sau khi thời gian sống của nó kết thúc, nhiễm sắc thể chết đi. Như vậy kích thước của quần thể sau một lần lặp là:

$$Popsize(t+1) = Popsize(t) + AuxPopsize(t) - D(t),$$

trong đó, D(t) là số nhiễm sắc thể chết đi ở thế hệ *t*.

Thủ tục GAVaPS

bắt đầu

t = 0

khởi tạo P(t)

lượng giá P(t)

khi (điều kiện-dừng chưa thỏa) **lặp**

bắt đầu

$t = t + 1$

tăng tuổi của mỗi cá thể lên 1

tái kết hợp P(t)

lượng giá P(t)

xóa tất cả các cá thể

có tuổi lớn hơn thời gian sống của nó khỏi P(t)

hết lặp

kết thúc

Hình P.3. Thuật giải GAVaPS

Có nhiều chiến lược để gán giá trị thời gian sống. Rõ ràng, khi gán một giá trị hằng số (lớn hơn 1) độc lập với bất cứ thống kê nào về tìm kiếm có thể làm tăng kích thước quần thể theo lũy thừa. Hơn nữa, do không có một cơ chế chọn lọc nào đúng nghĩa trong GAVaPS, nên không tồn tại áp lực chọn lọc nào, vì thế gán một giá trị hằng số cho tham số thời gian sống có thể đưa đến hiệu quả xấu cho thuật giải. Để đưa vào một áp lực chọn lọc, cần phải thực hiện việc tính toán thời gian sống tinh vi hơn. Các chiến lược tính toán thời gian sống cần phải củng cố các cá thể với độ thích nghi trên trung bình, và do đó, hạn chế các cá thể có độ thích nghi dưới trung bình và điều chỉnh kích thước của quần thể cho đúng với giai đoạn

tìm kiếm hiện hành (nhất là, ngăn tăng trưởng của quần thể theo lũy thừa và không hạ thấp các chi phí mô phỏng). Việc củng cố các cá thể thích nghi có thể dẫn đến việc phân phát con trong các quần thể phụ trợ. Do có xác suất bằng nhau để mỗi cá thể trải qua tái kết hợp di truyền, nên số con mong đợi của mỗi cá thể tỉ lệ với giá trị đời sống của nó (cho thời gian sống quyết định số thế hệ giữa cá thể trong quần thể). Vì thế các cá thể có giá trị thích nghi trên trung bình sẽ được hưởng các giá trị thời gian sống cao hơn. Khi tính thời gian sống, một trạng thái tìm kiếm sẽ được quan tâm. Do đó ta dùng một số phép đo trạng thái tìm kiếm: *AvgFit, MaxFit* và *MinFit* theo thứ tự biểu diễn các giá trị thích nghi trung bình, lớn nhất và nhỏ nhất trong quần thể hiện tại, và *AbsFitMax, AbsFitMin* cho các giá trị thích nghi lớn nhất và nhỏ nhất tìm được từ trước đến giờ. Cũng cần để ý rằng việc tính thời gian sống cần phải dễ thực hiện để không phí phạm tài nguyên.

Khi đã ghi nhận tất cả những điều trên, nhiều chiến lược tính thời gian sống đã được cài đặt và sử dụng cho các thử nghiệm. Tham số thời gian sống của cá thể thứ *i* (*lifetime[i]*) có thể xác định bằng cách:

(1) Phân phối theo tỉ lệ

$$\left(MinLT + \eta\frac{fitness[i]}{AvgFit}, MaxLT\right), \text{ fitness : độ thích nghi}$$

(2) Phân phối tuyến tính

$$MinLT + 2\eta\frac{fitness[i] - AbsFitMin}{AbsFitMax - AbsFitMin}$$

(3) Phân phối song tuyến tính

$$\begin{cases} MinLT + \eta\dfrac{fitness[i] - MinFit}{AvgFit - MinFit}, & (AvgFit \geq fitness[i]) \\ \dfrac{1}{2}(MinLT + MaxLT) \quad +\eta\dfrac{fitness[i] - AvgFit}{MaxFit - AvgFit}, & (AvgFit < fitness[i] \end{cases}$$

trong đó, *MaxLT* và *MinLT* lần lượt biểu diễn giá trị thời gian sống cực đại và cực tiểu cho phép (những giá trị này được cho như là các tham số GAVaPS), và $\eta=1/2(MaxLT-MinLT)$.

Chiến lược đầu tiên (phân phối theo tỉ lệ) xuất hiện từ ý tưởng về chọn bánh xe rulét: giá trị đời sống của một cá thể tỉ lệ với độ thích nghi của nó (trong các giới hạn *MinLT* và *MaxLT*). Nhưng chiến lược này có một khuyết điểm nghiêm trọng - nó không sử dụng thông tin nào về "những điều tốt khách quan" của cá thể tính được bằng cách liên hệ độ thích nghi của nó với giá trị tốt nhất từ trước. Quan sát này gợi ý chiến lược tuyến tính. Trong chiến lược này, giá trị thời gian sống được tính theo thích nghi cá thể liên hệ với giá trị tốt nhất hiện tại. Nhưng, nếu nhiều cá thể có độ thích nghi của chúng bằng hay xấp xỉ với giá trị tốt nhất, chiến lược đó dẫn đến việc phân phối các giá trị thời gian sống dài, như vậy sẽ làm tăng kích thước quần thể. Cuối cùng chiến lược song tuyến tính có khuynh hướng thỏa hiệp giữa hai chiến lược trước. Nó làm rõ những khác biệt giữa giá trị thời gian sống của những cá thể gần-như-tốt-nhất bằng thông tin về độ thích nghi trung bình, nhưng cũng quan tâm đến các giá trị thích nghi lớn nhất và nhỏ nhất tìm được từ trước.

Thuật giải GAVaPS được thử nghiệm trên những hàm sau:

G1: $-x\sin(10\pi x)+1$ $\quad -2.0\le x\le 1.0$

G2: $\text{integer}(8x)/8$ $\quad 0.0\le x\le 1.0$

G3: $x*\text{sgn}(x)$ $\quad -1.0\le x<=2.0$

G4: $0.5+\dfrac{\sin^2\sqrt{x^2+y^2}-0.5}{(1+0.001(x^2+y^2))^2}$ $\quad -100\le x,y\le 100$

Hàm G1 và G4 là các hàm có nhiều cực đại cục bộ. Hàm G2 không thể tối ưu hóa bằng bất kỳ kỹ thuật gradient, vì không có sẵn thông tin gradient nào. Hàm G3 biểu diễn "bài toán lừa". Khi cực đại

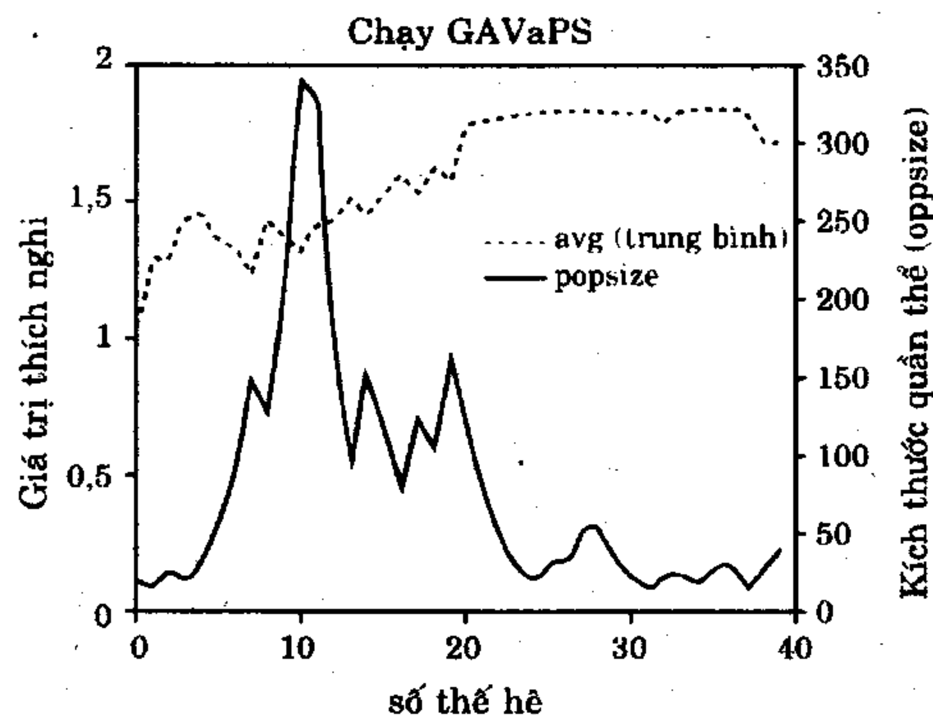
hóa hàm đó, hai hướng tăng trưởng dễ dàng được nhận ra, nhưng các biên được chọn sao cho cực đại cục bộ chỉ đạt được đối với một biên mà thôi. Trong trường hợp các kỹ thuật dựa trên gradient với cách tạo mẫu ngẫu nhiên, điều này có thể dẫn đến việc tìm thấy các cực đại cục bộ.

Hiệu quả GAVaPS được kiểm tra và so sánh với hiệu quả của thuật giải di truyền đơn giản của Golberg (SGA). Các phương pháp mã hóa bài toán cũng như các toán tử di truyền cũng tương tự đối với SGA và GAVaPS (một các mã hóa nhị phân đơn giản đã được dùng và hai toán tử di truyền: đột biến và lai tạo một điểm).

Đối với các thử nghiệm, ta đã có những giả định sau đây. Kích thước khởi tạo của một quần thể là 20. Trường hợp của SGA, kích thước của quần thể đầu tiên vẫn không đổi qua toàn bộ tiến trình. Tỉ lệ sinh sản ρ được đặt là 0.4 (tham số này vô nghĩa đối với SGA). Tỉ lệ đột biến được đặt là 0.015, và tỉ lệ lai tạo được đặt là 0.65. Chiều dài nhiễm sắc thể là 20. Qua tất cả các thử nghiệm, ta giả định rằng các giá trị thời gian sống cực đại và cực tiểu không đổi và bằng MaxLT=7 và MinLT=1.

Để so sánh SGA và GAVaPS, ta chọn hai tham số: chi phí của thuật giải được biểu diễn bằng *evalnum* (trung bình của số lượng giá hàm trên tất cả các lần chạy) và việc thực hiện được biểu diễn bằng *avgmax* (trung bình của các giá trị cực đại tìm được qua tất cả các lần chạy). Cả hai thuật giải đều có cùng điều kiện dừng: chúng chấm dứt khi không có tiến bộ nào theo nghĩa của giá trị tốt nhất tìm được cho thế hệ tiếp sau, *conv =20*. Quần thể được khởi tạo ngẫu nhiên, có 20 lần chạy độc lập được thực hiện. Rồi, những phép đo về hiệu quả và chi phí được tính trung bình trên 20 lần chạy này sẽ cho kết quả báo cáo. Trong khi thử nghiệm ảnh hưởng của một tham số trên hiệu quả và chi phí, các giá trị của các tham số được báo cáo trên đây vẫn không đổi, ngoại trừ giá trị đã được thử nghiệm.





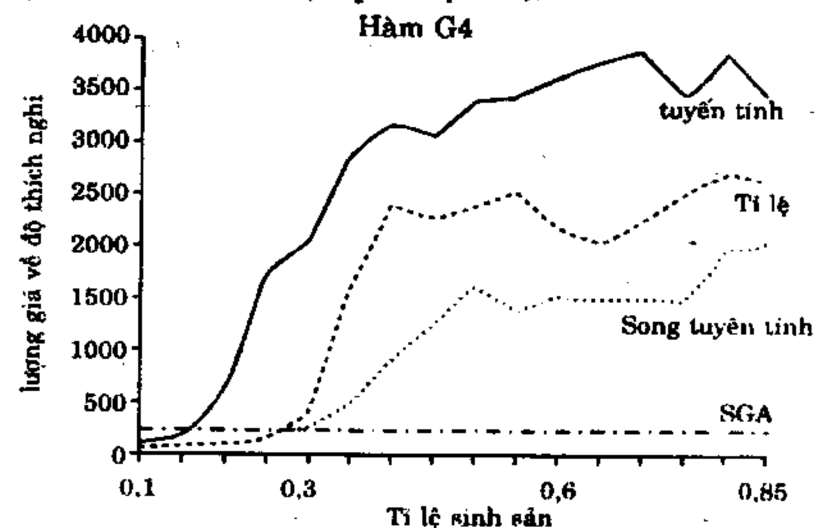
Hình P.4. PopSize(t) và thích nghi trung bình của quần thể đối với lần chạy GAVaPS.

Hình P.4 minh hoạ *PopSize*(t) và độ thích nghi trung bình của quần thể cho một lần chạy GAVaPS đối với hàm G4 có tính toán thời gian sống song tuyến tính (những quan sát tương tự cũng được thực hiện cho những hàm khác và những chiến lược khác cũng được thực hiện để phân phối các giá trị thời gian sống). Dạng của đường cong *PopSize*(t) có vẻ rất thú vị. Trước tiên, khi biến thể của sự thích nghi tương đối cao, kích thước quần thể tăng. Điều này có nghĩa là, GAVaPS thực hiện việc tìm kiếm tối ưu trải rộng. Khi lân cận của tối ưu được định vị, thuật giải bắt đầu hội tụ và kích thước quần thể giảm. Nhưng, vẫn còn tìm thêm cải thiện. Khi xuất hiện xác suất cho một kết quả tốt hơn, một sự bùng nổ khác về quần thể xảy ra, theo sau bởi một giai đoạn hội tụ khác. Dường như GAVaPS kết hợp tiến trình *tự-điều chỉnh* tốt bằng việc chọn kích thước quần thể tại mỗi giai đoạn của tiến trình tiến hóa.

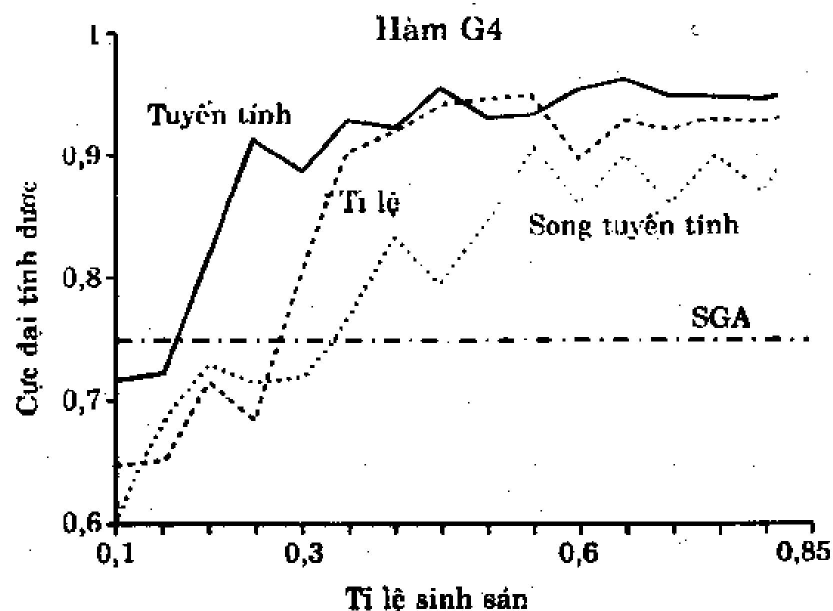


Hình P.5, P.6 cho thấy tác động của tỉ lệ sinh sản đối với hiệu quả của GAVaPS, đối với SGA, giá trị này không có ý nghĩa (do trong trường hợp này, quần thể cũ chồng lên quần thể mới). Trong trường hợp GAVaPS, giá trị này ảnh hưởng mạnh mẽ đến chi phí mô phỏng, chi phí này giảm bởi việc hạ thấp tốc độ sinh sản, nhưng không mất sự chính xác (xem các giá trị có liên quan *avgmax*). Từ những thử nghiệm này, có thể phán đoán dường như chọn lọc tối ưu của ρ xấp xỉ 0.4.

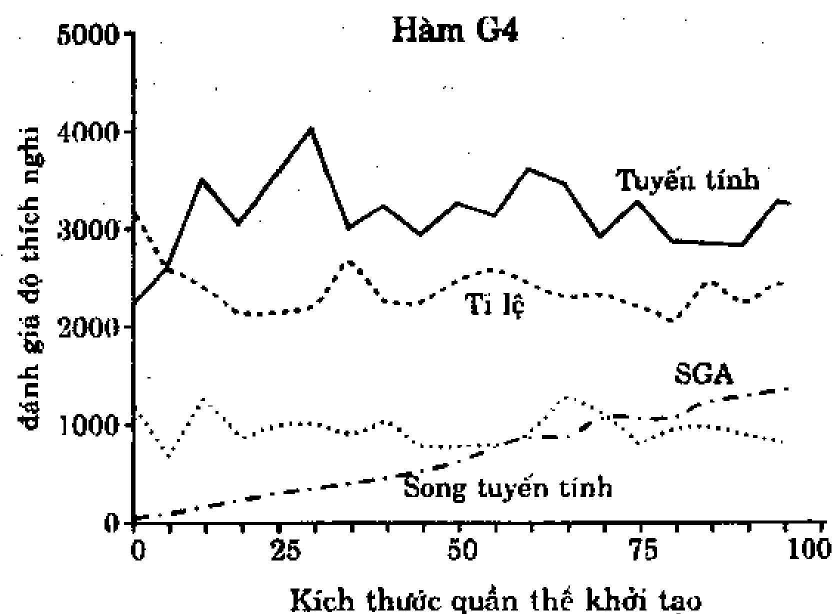
Hình P.7, P.8 cho thấy tác động của kích thước quần thể lúc khởi tạo đối chiếu với hiệu quả (*avgmax*) và chi phí tính toán (*evalnum*) của các thuật giải. Trong trường hợp SGA kích thước quần thể cho tất cả lần chạy không đổi và bằng giá trị ban đầu của nó. Như ta mong đợi, đối với SGA các giá trị thấp của kích thước quần thể mang ý nghĩa chi phí thấp và hiệu quả kém. Tăng kích thước quần thể lúc đầu sẽ cải thiện hiệu quả nhưng cũng tăng chi phí tính toán. Rồi đến giai đoạn "bão hòa hiệu quả" trong khi chi phí vẫn tăng tuyến tính. Trường hợp GAVaPS, kích thước quần thể khởi tạo không ảnh hưởng gì đến hiệu quả (rất tốt) cũng như chi phí (chấp nhận được và đủ đối với hiệu quả thật tốt).



Hình P.5. So sánh SGA và GAVaPS: Tỉ lệ sinh sản đối chiếu với số lượng giá

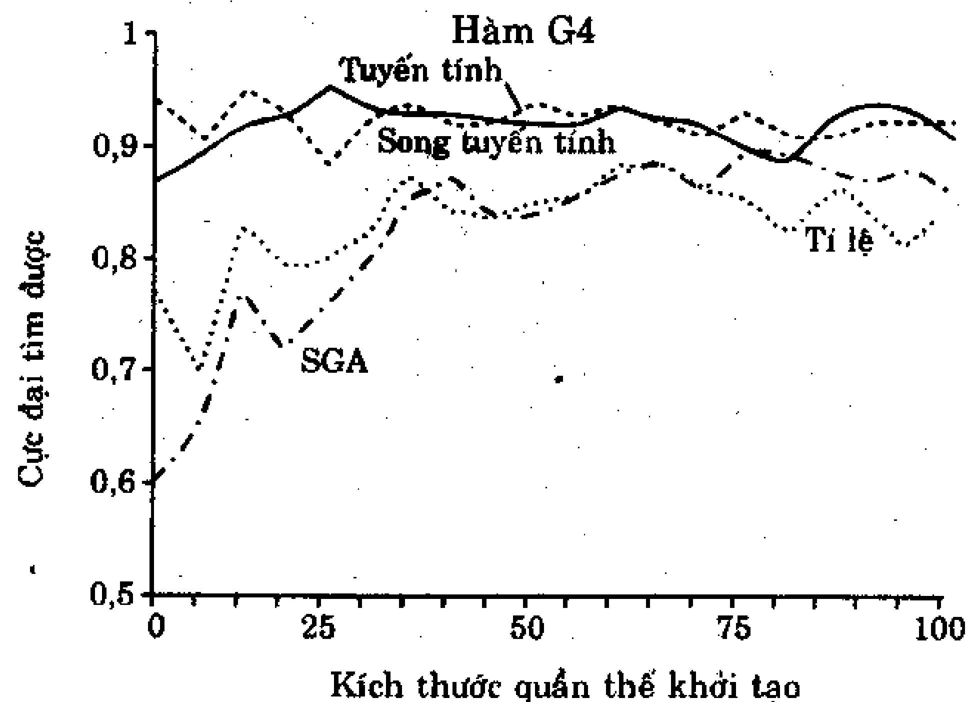


Hình P.6. So sánh SGA và GAVaPS: Tỉ lệ sinh sản đối chiếu với hiệu quả trung bình



Hình P.7. So sánh SGA và GAVaPS:kích thước quần thể lúc khởi tạo đối chiếu với số tiến hóa

Có thể thực hiện những quan sát tương tự bằng cách phân tích chi phí và hiệu quả của cả hai thuật giải trên các hàm còn lại G1-G3. Nhưng cần để ý rằng SGA có cách ứng xử tối ưu (hiệu quả tốt nhất và chi phí tối thiểu) đối với các giá trị kích thước quần thể khác nhau cho cả 4 bài toán. Mặt khác, GAVaPS chấp nhận một kích thước quần thể cho bài toán sắp tới và trạng thái tìm kiếm. Trong bảng sau đây, chúng ta báo cáo về hiệu quả và chi phí mô phỏng đạt được từ các kinh nghiệm với mọi hàm thử nghiệm G1-G4. Các hàng 'SGA', 'GAVaPS(1)', 'GAVaPS(2)', và 'GAVaPS(30)' chứa các giá trị tốt nhất tìm được (V) và số các tiến hóa hàm (E) của SGA và GAVaPS theo thứ tự của các phân phối tỉ lệ, tuyến tính, song tuyến tính. Các kích thước quần thể tối ưu đối với SGA trong các trường hợp thử nghiệm G1-G4 theo thứ tự là 75, 15, 75, 100.



Hình P.8. So sánh SGA và GAVaPS: đối chiếu kích thước quần thể với hiệu quả trung bình.

Bảng P.1. So sánh giữa ba chiến lược

Kiểu giải thuật	Hàm							
	G1		G2		G3		G4	
	V	E	V	E	V	E	V	E
SGA	2.814	1467	0.875	345	1.996	1420	0.969	2186
GAVaPS(1)	2.831	1708	0.875	970	1.999	1682	0.969	2133
GAVaPS(2)	2.841	3040	0.875	1450	1.999	2813	0.970	3739
GAVaPS(3)	2.813	1538	0.875	670	1.999	1555	0.972	2106

Chiến lược tuyến tính (2) được đặc trưng bằng hiệu quả tốt nhất và (thật không may) chi phí cao nhất. Mặt khác, chiến lược song tuyến tính (3) thì rẻ nhất, nhưng hiệu quả không tốt như tuyến tính. Cuối cùng, chiến lược tỉ lệ (1) cho ta một hiệu quả trung bình và chi phí trung bình. Nên lưu ý rằng, thuật giải GAVaPS với chiến lược phân phát thời gian sống (1) - (3), trong hầu hết các trường hợp, cho hiệu quả tốt hơn SGA. Chi phí cho GAVaPS (so với SGA) cao hơn, tuy nhiên các kết quả của SGA đã được báo cáo về vấn đề kích thước tối ưu của quần thể. Thí dụ, nếu kích thước quần thể đối với SGA rút kinh nghiệm qua các hàm G2 là 75 (thay vì tối ưu là 15), thì chi phí mô phỏng SGA sẽ là 1035.

Tri thức về sự chọn lọc các tham số GA cho thích hợp vẫn còn rời rạc và có nền tảng thực nghiệm. Trong những tham số này, kích thước quần thể dường như là tham số quan trọng nhất, vì nó ảnh hưởng mạnh mẽ đến chi phí mô phỏng GA. Cách tốt nhất đối với giá

trị thiết lập của nó là để nó tự tìm, dựa theo các nhu cầu thực tế của GA. Đó là ý tưởng ẩn sau phương pháp GAVaPS: ở các giai đoạn tìm kiếm khác nhau xử lý các kích thước quần thể khác nhau có lẽ là tốt nhất. Tuy nhiên, các kết quả đã được báo cáo chỉ là khởi đầu và việc phân phối tham số thời gian sống cần được nghiên cứu thêm.

5. Thuật giải di truyền, các ràng buộc và bài toán ba lô

Như đã nói trong phần dẫn nhập, các kỹ thuật xử lý ràng buộc của thuật giải di truyền có thể được gom vào một số loại. Một cách để xử lý với các ứng viên vi phạm ràng buộc là phát sinh những lời mà không xét đến ràng buộc rồi thưởng phạt chúng bằng cách giảm "độ tốt" của hàm lượng giá. Nói cách khác, bài toán ràng buộc được biến đổi thành không ràng buộc, bằng cách thêm thưởng phạt vào tất cả vi phạm ràng buộc; thưởng phạt này được gộp vào lượng giá hàm. Đương nhiên, có nhiều hàm thưởng phạt khác nhau có thể áp dụng. Một số hàm thưởng phạt gán một hằng số làm mức độ thưởng phạt. Những hàm thưởng phạt khác phụ thuộc vào mức độ vi phạm: vi phạm càng nhiều thì phạt càng nặng, (nhưng, tăng trưởng hàm có thể là logarít, tuyến tính, bình phương, lũy thừa v.v phù hợp với ràng buộc bị vi phạm).

Phiên bản bổ sung của các thưởng phạt là loại trừ các lời giải không-khả thi khỏi quần thể (nghĩa là, áp dụng thưởng phạt nghiêm khắc nhất: phạt chết. Kỹ thuật này được dùng thành công trong các chiến lược tiến hóa (chương 8) cho những bài toán tối ưu số. Nhưng, cách này có một số khuyết điểm. Đối với vài bài toán xác suất phát sinh (bằng phương tiện của các toán tử di truyền chuẩn) một lời giải khả thi khá nhỏ và thuật giải tốn một lượng lớn thời gian đánh giá các cá thể không hợp lệ. Hơn nữa, trong cách này, những lời giải không khả thi không đóng góp vào vùng lưu trữ gen của các quần thể.

Một loại khác của phương pháp xử lý ràng buộc là dựa trên việc ứng dụng các thuật giải sửa chữa đặc biệt để "sửa" những lời giải không khả thi phát sinh như thế. Lần nữa, những thuật giải sửa chữa như thế có thể quá căng về mặt tính toán khi chạy và thuật giải có được phải được thiết kế theo mẫu của ứng dụng cụ thể. Hơn nữa, đối với một số bài toán, tiến trình sửa chữa có lẽ cũng khó ngang với bài toán gốc.

Cách tiếp cận thứ ba tập trung vào việc sử dụng các ánh xạ biểu diễn đặc biệt (các bộ giải mã) bảo đảm (hoặc ít nhất tăng xác suất của) thế hệ của lời giải khả thi hay việc dùng những toán tử bài toán-đặc tả bảo toàn tính khả thi của các lời giải. Nhưng chạy những bộ giải mã bằng máy tính là việc rất căng. Tuy vậy, tất cả các ràng buộc có thể dễ dàng cài đặt theo cách này và thuật giải có được phải được thiết kế theo mẫu ứng dụng cụ thể.

Trong phần này, ta khảo sát những kỹ thuật trên với một bài toán cụ thể: bài toán balô 0/1. Bài toán này dễ dàng hình thức hóa, nhưng phiên bản quyết định của nó thuộc về họ các bài toán NP-đầy đủ. Đây là bài tập thú vị để lượng giá những thuận lợi và bất lợi của các tất cả xử lý ràng buộc trên bài toán cụ thể có một ràng buộc duy nhất này: các kết luận của nó có thể áp dụng cho nhiều bài toán tối ưu hóa có ràng buộc. Nhưng cũng nên chú ý rằng mục đích chính của phần này là minh họa khái niệm về các bộ giải mã, thuật giải di truyền, và các hàm thưởng phạt (được nói ngắn gọn trong phần dẫn nhập) trên một thí dụ cụ thể; nó không hề là một nghiên cứu đầy đủ về các phương pháp có thể có. Vì lý do đó ta không cung cấp các lời giải tối ưu cho các trường hợp thử nghiệm: ta chỉ đưa ra một vài so sánh giữa các phương pháp được giới thiệu.

5.1. Bài toán ba lô 0/1 và dữ liệu thử nghiệm

Có nhiều loại bài toán kiểu ba lô khác nhau, trong đó cho trước một tập thực thể, cùng các giá trị và kích thước của chúng và ta mong chọn được một hay nhiều tập con rời nhau sao cho tổng kích thước trong mỗi tập con không vượt quá giới hạn cho phép và tổng giá trị đã chọn được là lớn nhất. Bài toán này là một thí dụ về bài toán thuộc lớp NP-khó-và-lớn có thể tiếp cận bằng cách dùng heuristic. Bài toán được chọn cho thử nghiệm này là bài toán ba lô 0/1. Công việc phải làm là, đối với một tập các trọng số đã cho $W[i]$, lợi ích $P[i]$ và sức chứa C, tìm một vectơ nhị phân $x = <x[1],\ldots, x[n]>$, sao cho:

$$\sum_{i=1}^{n} x[i].W[i] \le C$$

và để cho:

$$P(x) = \sum_{i=1}^{n} x[i].P[i]$$

là cực đại.

Khó khăn của những bài toán loại này chịu ảnh hưởng rất lớn bởi tương quan giữa lợi ích và các trọng, nên ta khảo sát ba tập dữ liệu thử:

- *Không tương quan*

  $W[i]$:= random ([1..v]) và

  $P[i]$:= random ([1..v])

- *Tương quan yếu*

  $W[i]$:= random ([1..v]) và

  $P[i]$:= $W[i]$ + random ([-r..r]),

  (nếu với một i bất kỳ, $P[i] \le 0$, giá trị đó bị bỏ qua và phép tính được lặp lại cho đến khi $P[i]>0$).

- *Tương quan mạnh*

  $W[i]$:= random ([1..v]) và

  $P[i]$:= $W[i]+r$

  Tương quan cao hơn có nghĩa là giá trị khác biệt nhỏ hơn:

  $max_{i=1..n}[P[i]/W[i]] - min_{i=1..n}[P[i]/W[i]]$;

  (những bài toán tương quan cao hơn có mức khó khăn cao hơn).

  Dữ liệu được phát sinh bằng các thiết lập tham số: $v=10$ và $r=5$. Đối với các thử nghiệm ta đã dùng ba tập dữ liệu cho mỗi thực thể lần lượt có n = 100, 250, và 500 món. Ta quan tâm đến hai loại ba lô:

- *sức chứa ba lô hạn chế*

  Ba lô có sức chứa $C_1 = 2v$. Trong trường hợp này, lời giải tối ưu chứa rất ít mục. Đối với những điều kiện không đầy đủ, một vùng hầu như phụ trách toàn bộ miền.

- sức chứa ba lô trung bình

  Ba lô có sức chứa $C_2 = 0.5\sum_{i=1}^{n} W[i]$. Trong trường hợp này khoảng phân nửa các món nằm trong lời giải tối ưu.

Việc tăng sức chứa C không làm tăng nhiều thời gian tính toán của các thuật giải cổ điển.

5.2. Mô tả các thuật giải

Ba loại thuật giải được cài đặt và thử nghiệm là: thuật giải dựa trên hàm thưởng phạt ($A_p[i]$, i là chỉ số của một thuật giải cụ thể trong lớp này), thuật giải dựa trên phương pháp sửa chữa ($A_r[i]$),

thuật giải dựa trên việc giải mã ($A_d[i]$). Ta lần lượt bàn về ba loại thuật giải này:

THUẬT GIẢI $A_P[I]$

Trong tất cả các thuật giải loại này, một chuỗi nhị phân có chiều dài n biểu diễn lời giải x của bài toán: món thứ i được chọn cho ba lô nếu và chỉ nếu $x[i] = 1$.

Độ thích nghi *eval (x)* của mỗi chuỗi được định nghĩa là:

$$eval(x) = \sum_{i=1}^{n} x[i].P[i] - Pen(x),$$

trong đó, hàm thưởng phạt *Pen(x)* sẽ bằng 0 với lời giải khả thi x, nghĩa là, các lời giải thoả $\sum_{i=1}^{n} x[i].W[i] \le C$, ngược lại, *Pen(x)* lớn hơn 0.

Có nhiều chiến lược để gán giá trị thưởng phạt. Ở đây, ta chỉ quan tâm đến ba trường hợp mà hàm thưởng phạt tăng theo lôgarít, tuyến tính và bình phương tùy theo mức độ xâm phạm:

$A_p[1]$: $Pen(x) = \log_2(1 + \rho.(\sum_{i=1}^{n} x[i].W[i] - C))$

$A_p[2]$: $Pen(x) = \rho.(\sum_{i=1}^{n} x[i].W[i] - C)$

$A_p[3]$: $Pen(x) = (\rho.(\sum_{i=1}^{n} x[i].W[i] - C))^2$

Trong cả ba trường hợp, $\rho = max_{i=1..n}\{P[i]/W[i]\}$.

THUẬT GIẢI $A_R[1]$

Cũng như trong loại thuật giải trước, một chuỗi nhị phân có chiều dài *n* biểu diễn lời giải *x* của bài toán: món thứ *i* được chọn cho ba lô nếu và chỉ nếu $x[i] = 1$.

Độ thích nghi *eval (x)* của mỗi chuỗi được định nghĩa là,

$$eval\ (x) = \sum_{i=1}^{n} x'[i].P[i]$$

trong đó, vectơ *x'* là phiên bản được sửa chữa của vectơ gốc *x*.

Ở đây có 2 khía cạnh thú vị. Trước tiên, ta có thể có nhiều phương pháp sửa chữa khác nhau. Thứ hai, vài phần trăm các nhiễm sắc thể được sửa chữa có thể thay cho các nhiễm sắc thể gốc trong quần thể. Tỉ lệ thay thế đó thay đổi từ 0% đến 1%, hay theo *luật 5%*. Luật này cho rằng, trong thực nghiệm, nếu thay các nhiễm sắc thể gốc với xác suất 5%, hiệu quả của thuật giải sẽ tốt hơn khi thay thế bằng các tỉ lệ khác (nhất là, nó tốt hơn các chiến lược 'không bao giờ thay' hoặc ' luôn luôn thay').

Ta cài đặt và thử nghiệm hai thuật giải sửa chữa khác nhau, cả hai đều dựa trên cùng một thủ tục như trong hình P.9.

Thủ tục sửa chữa (*x*)

Bắt đầu

ba lô quá đầy:= false

$x' := x$

nếu $\sum_{i=1}^{n} x'[i] * W[i] > C$

thì ba lô - quá đầy:= true

khi (ba lô - quá đầy) **làm**

bắt đầu

i: = **chọn** một món trong ba lô

loại món đó khỏi ba lô, nghĩa là, x'[i]:= 0

nếu $\sum_{i=1}^{n} x'[i] * W[i] \le C$

thì ba lô-quá đầy:= false

hết lặp

Kết thúc

Hình P.9. Thủ tục sửa chữa

Hai thuật giải sửa chữa như trình bày ở trên chỉ khác nhau trong câu lệnh **chọn** của thủ tục chọn lọc, lệnh này chọn một món để lấy ra khỏi ba lô:

- $A_r[1]$ (sửa chữa ngẫu nhiên) thủ tục **chọn lọc** chọn ngẫu nhiên một phần tử trong ba lô.

- $A_r[2]$ (sửa chữa tham lam) Tất cả các món trong ba lô được sắp theo thứ tự giảm dần về lợi ích của chúng đối với tỉ lệ trọng số. Thủ tục chọn lọc luôn luôn chọn món cuối cùng (trong danh sách các món có sẵn) để xóa bỏ.

THUẬT GIẢI $A_d[I]$

Một bộ giải mã cho bài toán ba lô dựa trên biểu diễn nguyên (integer). Ở đây ta dùng biểu diễn thứ tự các món được chọn. Mỗi nhiễm sắc thể là một vectơ có *n* số nguyên; thành phần thứ *i* của vectơ là một số nguyên trong khoảng từ 1 đến $n\text{-}i\text{+}1$. Biểu diễn thứ tự tham chiếu đến danh sách *L* các món; một vectơ được giải mã bằng cách chọn món thích hợp trong danh sách hiện hành. Thí dụ, đối với danh sách *L=(1,2,3,4,5,6)*, vectơ <4,3,4,1,1,1> được giải mã thành chuỗi các mục sau đây: 4, 3, 6 (do 6 là phần tử thứ tư trong danh sách hiện hành sau khi loại bỏ 4 và 3), 1, 2 và 5. Hiển nhiên, trong phương pháp này một nhiễm sắc thể có thể được thông dịch là chiến lược kết hợp các món vào lời giải. Hơn nữa, lai một - điểm được áp dụng cho hai cha-mẹ hợp lệ bất kỳ, cũng sẽ cho ra một con hợp lệ. Toán tử đột biến được định nghĩa theo cùng cách với biểu diễn nhị phân: nếu gen thứ *i* bị đột biến, nó sẽ có một giá trị ngẫu nhiên trong khoảng $[1..n\text{-}i\text{+}1]$. Thuật giải giải mã được trình bày trong hình P10.

Hai thuật giải dựa trên các kỹ thuật giải mã được bàn ở đây chỉ khác nhau ở thủ tục **xây dựng**:

- $A_d[1]$ (giải mã ngẫu nhiên). Trong thuật giải này, thủ tục xây dựng tạo danh sách *L* sao cho thứ tự các món trong danh sách tương ứng với các món trong tập tin nhập (ngẫu nhiên).

- $A_d[2]$ (giải mã tham lam). Thủ tục xây dựng tạo danh sách L theo thứ tự giảm lợi tức theo tỉ lệ trọng số. Việc giải mã vectơ *x* được thực hiện trên cơ sở thứ tự sắp xếp (có một số điểm tương tự với phương pháp $A_r[2]$). Thí dụ, $x[i]$ = 23 được thông dịch thành món thứ 23 (theo thứ tự giảm lợi tức đối với tỉ lệ trọng số trên danh sách hiện hành L.

4.5.3. Thử nghiệm và kết quả

Trong tất cả các thử nghiệm, kích thước kết quả là hằng số và bằng 100. Cũng thế, các xác suất đột biến và lai không đổi, và lần lượt là: 0.05 và 0.65.

Thủ tục giải mã (x)

Bắt đầu

xây dựng danh sách L

i:= 1

TổngTrọng := 0

TổngLợiÍch:= 0

khi $i \le$ n **làm**

bắt đầu

j:= x[i]

xóa món thứ j khỏi danh sách L

nếu *TổngTrọng + W[j]* ≤ C **thì**

bắt đầu

TổngTrọng:= TổngTrọng +Trọng [j]

TổngLợiÍch:= TổngLợiÍch + Lợi Ích[j]

Hết nếu

i:= i+1

Hết lặp

Kết thúc

Hình P.10. Thủ tục giải mã của biểu diễn thứ tự

Khi ta đã gom lời giải tốt nhất tìm được trong vòng 500 thế hệ như là độ đo hiệu quả của thuật giải. Thực nghiệm chứng minh rằng, sau một số thế hệ như vậy, ta không thấy thêm cải thiện nào. Các kết quả được trình bày trong bảng 2 là các giá trị trung bình của 25 thử nghiệm. Các lời giải chính xác không được liệt kệ ở đây; bảng chỉ so sánh hiệu quả tương đối giữa các thuật giải khác nhau. Lưu ý, các tập tin dữ liệu không được sắp xếp (thứ tự của các món là tùy ý, không liên quan đến các tỉ lệ *P[i]/W[i]* của chúng). Các thực thể với

sức chứa C_1 và C_2 theo thứ tự không chế các sức chứa nhỏ và trung bình.

Kết quả của các phương pháp $A_r[1]$ và $A_r[2]$ đã đạt được bằng cách sử dụng luật 5%. Ta đã xem xét luật 5% làm việc với bài toán ba lô 0/1 (luật được phát hiện từ thực nghiệm trên hai bài toán tổ hợp khác: bài toán thiết kế mạng và bài toán tô màu bản đồ). Với món đích so sánh, ta đã chọn các tập dữ liệu thử nghiệm có tương quan yếu giữa giá trị và lợi ích. Tất cả các giá trị tham số đều cố định và tỉ lệ sửa chữa thay đổi từ 0% đến 100%. ta không thấy ảnh hưởng của luật 5% trên hiệu quả của thuật giải di truyền. Các kết quả (thuật giải $A_r[2]$) được tập hợp trong bảng 3.

Ta có thể tóm tắt những kết luận chính rút ra từ các thử nghiệm như sau:

- Các hàm thưởng phạt $A_p[i]$ (với mọi *i*) không tạo ra kết quả khả thi với những bài toán có sức chứa ba lô nhỏ (C_1). Đây là trường hợp đối với bất cứ số món nào (*n*=100, 250, và 500) và với mọi tương quan.
- Chỉ phán đoán từ những kết quả thử nghiệm trên các bài toán có sức chứa ba lô trung bình (C_2), thuật giải $A_p[1]$ dựa trên hàm thưởng phạt lôgarit thành công rõ rệt: nó hiệu quả hơn tất cả các kỹ thuật khác qua mọi trường hợp (không tương quan, tương quan yếu và mạnh, với *n*= 100, 250, và 500 món). Nhưng, như đã trình bày, nó lại thất bại ở những bài toán có sức chứa hạn chế.
- Chỉ phán đoán từ những kết quả thử nghiệm trên các bài toán sức chứa ba lô nhỏ (C_1), phương pháp sửa chữa $A_r[2]$ (sửa chữa tham lam) cho thấy: nó hiệu quả hơn tất cả các phương pháp khác qua mọi thử nghiệm.

tương quan	số món	thực thể	Phương pháp						
			$A_p[1]$	$A_p[2]$	$A_p[3]$	$A_r[1]$	$A_r[2]$	$A_d[1]$	$A_d[2]$
không	100	C1	*	*	*	62.9	94.0	63.5	59.4
		C2	398.1	341.3	342.6	344.6	371.3	354.7	353.3
	250	C1	*	*	*	62.6	135.1	58.0	60.4
		C2	919.6	837.3	825.5	842.2	894.4	867.4	857.5
	500	C1	*	*	*	63.9	156.2	61.0	61.4
		C2	1712.2	1570.8	1565.1	1577.4	1663.2	1602.8	1597.0
yếu	100	C1	*	*	*	39.7	51.0	38.2	38.4
		C2	408.5	327.0	328.3	30.1	358.2	333.6	332.3
	250	C1	*	*	*	43.7	74.0	42.7	44.7
		C2	920.8	791.3	788.5	798.4	852.1	804.4	799.0
	500	C1	*	*	*	44.5	93.8	43.2	44.5
		C2	1729.0	1531.8	1532.0	1538.6	1624.8	1548.4	1547.0
mạnh	100	C1	*	*	*	61.6	90.0	59.5	59.5
		C2	741.7	564.5	564.4	566.5	577.0	576.2	576.2
	250	C1	*	*	*	65.5	117.0	65.5	64.0
		C2	1631.9	1339.5	1343.4	1345.8	1364.4	1366.4	1359.0
	500	C1	*	*	*	67.5	120.0	67.1	64.1
		C2	3051.6	2703.8	2700.8	2709.5	2748.1	2738.0	2744.0

Bảng P.2. Kết quả các thử nghiệm; dấu sao có nghĩa là không tìm được lời giải hợp lệ thỏa ràng buộc thời gian được cho.

Những kết quả này cũng dễ hiểu. Trong trường hợp sức chứa ba lô nhỏ, chỉ có một phần nhỏ các tập con các món có thể đóng góp lời giải khả thi; do đó hầu hết các phương pháp thưởng phạt đều thất

bại. Đây cũng là trường hợp của những bài toán tổ hợp khác: tỉ lệ giữa phần khả thi của không gian tìm kiếm và của toàn bộ không gian tìm kiếm càng nhỏ, các phương pháp hàm thưởng phạt càng khó cung cấp những kết quả khả thi.

Ngược lại, thuật giải sửa chữa có hiệu quả hoàn toàn tốt. Trường hợp sức chứa ba lô trung bình, phương pháp hàm thưởng phạt lôgarít (nghĩa là thưởng phạt nhỏ) có ưu thế hơn: sẽ thú vị nếu để ý rằng kích thước của bài toán không ảnh hưởng đến các kết quả.

Như đã trình bày trước đây, những thử nghiệm này chưa cung cấp một bức tranh toàn diện; trong tương lai gần, người ta dự tính làm nhiều thử nghiệm bổ sung. Trong loại hàm thưởng phạt, sẽ thú vị để thử nghiệm với những công thức bổ sung. Tất cả những thưởng phạt được quan tâm đều thuộc dạng $Pen(\mathbf{x}) = f(\mathbf{x})$, trong đó, f là hàm lôgarít, tuyến tính hay bình phương. Một khả năng khác để thử nghiệm với $Pen(\mathbf{x}) = a+f(\mathbf{x})$ với một hằng số a nào đó. Như vậy, sẽ cung cấp một thưởng phạt tối thiểu cho bất kỳ một vectơ không khả thi nào. Hơn nữa, thử nghiệm với các hàm thưởng phạt *động* cũng thú vị, ở đây các giá trị của các hàm tùy thuộc vào các tham số bổ sung, như số thế hệ chẳng hạn. Cũng đáng để thử nghiệm với các hàm thưởng phạt *tự thích nghi*. Sau cùng, xác suất của các toán tử được áp dụng có thể thích nghi (như trong các chiến lược tiến hóa); một số thử nghiệm khởi tạo cho thấy các kích thước quần thể thích nghi có thể có một số thuận lợi (phần 4); vì thế, những ý kiến về các hàm thưởng phạt thích nghi cũng đáng được quan tâm. Trong phiên bản đơn giản nhất của nó, hệ số thưởng phạt có thể là một phần của vectơ lời giải và trải qua tất cả những thay đổi di truyền (ngẫu nhiên) (trái với ý kiến về các hàm thưởng phạt động, ở đây hệ số thưởng phạt như thế bị thay đổi trên cơ sở thường quy là hàm của số thế hệ, chẳng hạn).

Cũng có thể thử nghiệm với nhiều lược đồ sửa chữa, gồm có cả những heuristic khác với tỉ lệ của lợi ích và trọng số. Cũng sẽ thú vị

khi kết hợp các phương pháp thưởng phạt với các giá trị sửa chữa: những lời giải không khả thi của các thuật giải Ap[i] có thể được sửa chữa thành khả thi.

Trong loại bộ giải mã cần phải thử nghiệm với nhiều biểu diễn (integer) khác nhau (như đã được thực hiện cho bài toán người du lịch trước): biểu diễn kề (lai với các cạnh thay đổi, đột biến các phần của hành trình con, hay đột biến heuristic), hoặc biểu diễn đường dẫn với các đột biến PMX, OX và CX, hay ngay cả đột biến tái kết hợp cạnh). Cũng thú vị khi so sánh lợi ích của những biểu diễn này và các toán tử của bài toán ba lô 0/1 (như đã thực hiện đối với bài toán người du lịch và lập thời khóa biểu). Cũng hoàn toàn có khả năng là một đột biến mới nào đó cung cấp những kết quả tốt nhất.

6. NHỮNG Ý KIẾN KHÁC

Những phần trước trong phụ lục này, ta đã bàn một số vấn đề liên quan đến việc xóa lỗi có thể có trong các cơ chế tạo mẫu. Mục đích cơ bản của nghiên cứu này nhằm nâng cao chất lượng của tìm kiếm di truyền để tránh hội tụ sớm. Trong những năm gần đây, đã có một số nỗ lực nghiên cứu khác để xử lý lược đồ tốt hơn, nhưng sử dụng những phương pháp khác nhau. Trong phần này, ta bàn về những phương pháp đó.

Hướng thứ nhất có liên quan đến toán tử di truyền lai. Toán tử này được gợi ý từ một tiến trình sinh học; nhưng, nó có vài khuyết điểm. Thí dụ, giả sử có hai lược đồ hiệu quả cao:

S_1= (0 0 1 * * * * * * * * 0 1) và

S_2= (* * * * 1 1 * * * * * * *).

Cũng có hai chuỗi trong quần thể v_1 và v_2 lần lượt tương ứng với S_1 và S_2:

v_1 = (0010001101001)

v_2 = (1110110001000).

Rõ ràng lại không thể thực hiện một số kết hợp tính chất được mã hóa trên nhiễm sắc thể; không thể có một chuỗi phù hợp với lược đồ:

S_3 = (0 0 1 * 1 1 * * * * * 0 1),

bởi lược đồ thứ nhất sẽ bị hủy.

Ta cũng bàn thêm về phép lai một điểm cổ điển trong một số dạng không đối xứng giữa đột biến và lai: đột biến phụ thuộc chiều dài của nhiễm sắc thể còn lai thì không. Thí dụ, nếu xác suất của đột biến P_m là 0.01, còn chiều dài nhiễm sắc thể là 100, số bit đột biến mong đợi trong nhiễm sắc thể là một. Nếu chiều dài của nhiễm sắc thể là 1000, số bit đột biến mong đợi trong nhiễm sắc thể là mười. Mặt khác, trong cả hai trường hợp này, lai một điểm kết hợp hai chuỗi trên cơ sở của một vị trí lai, bất chấp chiều dài của chuỗi.

Một số nhà nghiên cứu đã thử nghiệm những phép lai khác nhau. Thí dụ, lai hai điểm chọn hai vị trí lai và nguyên liệu nhiễm sắc thể được hoán vị giữa chúng. Hiển nhiên, chuỗi v_1 và v_2 có thể sinh một cặp con:

v'_1 = (001|01100|01001) và

v'_2 = (111|00011|01000),

ở đây v'_1 phù hợp với:

S_3 = (0 0 1 * 1 1 * * * * * 0 1), không thể xảy ra với lai một điểm.

Tương tự, có những lược đồ mà lai hai điểm không thể kết hợp, ta cũng có thể thử nghiệm với lai nhiều điểm, một mở rộng tự nhiên của lai hai điểm. Nhưng chú ý rằng, do lai nhiều điểm phải xen kẽ các đoạn (có được sau khi cắt một nhiễm sắc thể thành *s* mảnh) giữa hai cha-mẹ, nên số đoạn phải chẵn, nghĩa là lai nhiều điểm không phải là mở rộng tự nhiên của lai một điểm.

Schaffer và Morishima đã thử nghiệm phép lai phân phối các điểm lai của nó bằng cùng những tiến trình có sẵn tại chỗ (sự tồn tại của cá thể thích nghi nhất và tái kết hợp). Điều này được thực hiện bằng cách đưa những điểm đặc biệt vào biểu diễn chuỗi, những điểm này kiểm soát những vị trí trong chuỗi nơi đã xảy ra lai. Hy vọng rằng những vị trí lai có thể trải qua một tiến trình tiến hóa: nếu một vị trí nào đó sinh các con yếu, vị trí đó sẽ chết (và ngược lại). Thực nghiệm cho thấy phép lai thích nghi có hiệu quả tốt bằng hoặc hơn GA cổ điển trên một số lớn các bài toán thử nghiệm. Spears đã thử nghiệm với một phép lai đặc biệt (hai phép lai, lai một điểm và lai đồng nhất, được xét đến) bằng cách mở rộng biểu diễn nhiễm sắc thể bằng bit bổ sung.

Một số nhà nghiên cứu đã thử nghiệm với các phép lai khác: lai *phân đoạn* và lai *trộn*. Lai phân đoạn là biến thể của lai nhiều điểm, cho phép số điểm lai thay đổi được. Số (cố định) điểm lai (hoặc cắt đoạn) được thay bằng tỉ lệ chuyển đổi đoạn. Tỉ lệ này đặc tả xác suất mà một đoạn sẽ chấm dứt tại một điểm nào đó trong chuỗi. Thí dụ, nếu tỉ lệ chuyển đổi đoạn là $s = 0.2$, thì nếu bắt đầu từ đầu đoạn, cơ hội chấm dứt đoạn này tại mỗi bit sẽ là 0.2. Nói cách khác, với tỉ lệ chuyển đổi đoạn $s = 0.2$, số đoạn mong đợi sẽ là $m/5$, nhưng không như lai nhiều điểm, số điểm lai sẽ thay đổi. Lai trộn có thể được xem là một cơ chế bổ sung có thể áp dụng với các cách lai khác. Nó độc lập với số điểm lai. Lai xáo trộn là phép lai (1) xáo trộn ngẫu nhiên các vị trí bit của hai chuỗi kế nhau, (2) bắt chéo các chuỗi, nghĩa là trao đổi các đoạn giữa các điểm lai, và (3) không xáo trộn các chuỗi.

Một bước tổng quát hóa xa hơn của lai một điểm, hai và nhiều điểm là lai đồng nhất: đối với mỗi bit trong đứa con đầu tiên, nó quyết định (với một xác suất p nào đó) cha-mẹ nào sẽ đóng góp giá trị của nó vào vị trí đó. Đứa con thứ hai sẽ nhận bit từ cha-mẹ khác.

Thí dụ, cho $p = 0.5$ (lai đồng nhất 0.5), các chuỗi:

$v_1 = (0010001101001)$ và

$v_2 = (1110110001000)$

có thể sinh ra cặp con sau đây:

$v'_1 = (0_1 0_1 1_2 0_1 1_2 1_2 0_2 1_1 0_1 1_2 0_1 0_1 0_2)$ và

$v'_2 = (1_2 1_2 1_1 0_2 0_1 0_1 1_1 0_2 0_2 1_1 0_2 0_2 1_1)$

Do lai đồng nhất trao đổi các bit chứ không phải các đoạn, nó có thể kết hợp các tính năng với nhau bất chấp vị trí tương đối của chúng. Đối với nhiều bài toán, khả năng này quan trọng hơn những bất lợi của việc phá hủy các khối kiến trúc gây ra. Nhưng, với những bài toán khác, lai đồng nhất lại kém hiệu quả hơn lai hai điểm. Syswerda so sánh về lý thuyết lai đồng nhất 0.5 với lai một điểm và hai điểm. Spears và De Jong đã phân tích về phép lai p-đồng nhất, nghĩa là phép lai có liên quan với các điểm lai $m*p$ trung bình.

Eshelman thử nghiệm nhiều toán tử lai khác nhau. Kết quả cho thấy lai một điểm đã thất bại; nhưng những toán tử lai khác cũng không thành công rõ ràng. Một ý kiến chung về những thử nghiệm trên là mỗi phép lai này chỉ đặc biệt có tốt trong số lớp bài toán và rất xấu trong những bài toán khác. Điều này càng củng cố ý kiến của chúng ta về sự phụ thuộc của các toán tử vào bài toán trong lập trình tiến hóa.

Muhlenbein và Voigt phát hiện một số thuộc tính của toán tử tái kết hợp mới, gọi là *tái kết hợp vùng chứa*, ở đây các gen được nhặt ra ngẫu nhiên từ vùng chứa được định nghĩa bởi các cha-mẹ được chọn. Một viễn cảnh thú vị của toán tử này là nó cho phép nhiều cha-mẹ tham gia vào việc sinh sản một con. Lai nhiều cha-mẹ như thế cũng được Eiben và một số tác giả khác đề cập, ở đây nhiều *kỹ thuật quét gen* (sinh một con duy nhất từ nhiều cha-mẹ) đã được nghiên cứu. Renders và Bersini đã thử nghiệm với phép lai một chiều trên những bài toán tối ưu số; phép lai này gồm việc tính trọng tâm của nhóm các cha-mẹ và gồm cả việc chuyển từ cá thể tệ nhất vượt cả điểm trọng tâm.

Nhiều nhà nghiên cứu cũng để ý vào ảnh hưởng của các tham số điều khiển của GA (kích thước quần thể, xác suất thi hành các toán tử) đối với hiệu quả của hệ thống. Grefenstette áp dụng một siêu-GA để điều khiển các tham số của một GA khác. Goldberg thì phân tích lý thuyết về kích thước quần thể tối ưu. Các kết quả gợi ý rằng (1) đột biến đóng một vai trò quan trọng, (2) tầm quan trọng của tỉ lệ lai lại nhỏ hơn mong đợi, (3) chiến lược tìm kiếm chỉ dựa trên chọn lọc và đột biến có thể là một thủ tục đột biến mạnh, ngay cả khi không có sự trợ giúp của lai. Nhưng GA vẫn thiếu các heuristic tốt để quyết định những giá trị tốt cho các tham số của chúng: không có thiết lập đơn lẻ nào có thể phổ dụng cho tất cả các bài toán được bàn đến. Dường như việc tìm các giá trị tốt cho các tham số GA mang tính nghệ thuật nhiều hơn là khoa học.

Cho đến lúc này, ta cũng chỉ bàn đến hai toán tử di truyền cơ bản: lai (một điển, hai điểm, đồng nhất, vv...) và đột biến được áp dụng cho những cá thể hay những bit với một số tỉ lệ cố định (các xác suất của lai p_c và đột biến p_m). Như ta đã thấy trong chương trình minh hoạ ở chương 2, có thể áp dụng phép lai và đột biến vào cùng một cá thể (thí dụ v_{13}). Thật ra, hai toán tử cơ bản này được xem như một toán tử tái kết hợp, toán tử "lai và đột biến", vì cả hai đều áp dụng được vào các cá thể một cách đồng thời. Khi thực

nghiệm với các toán tử di truyền, có một khả năng làm cho chúng độc lập: một (hoặc nhiều) toán tử này sẽ được áp dụng suốt giai đoạn sinh sản, nhưng không phải cả hai. Có một vài lợi ích trong cách phân chia như vậy. Đầu tiên, đột biến sẽ không được áp dụng lâu hơn vào kết quả của toán tử lai, làm cho toàn thể tiến trình đơn giản hơn một cách có mục đích. Kế đến, mở rộng một danh sách các toán tử di truyền bằng cách thêm vào các toán tử mới sẽ dễ hơn: danh sách như vậy có thể chứa nhiều toán tử phụ thuộc bài toán. Đây chính là ý kiến ẩn sau lập trình tiến hóa: có nhiều "toán tử di truyền" phụ thuộc bài toán, được áp dụng vào các cấu trúc dữ liệu cá thể. Cũng nên nhớ rằng, đối với các chương trình tiến hóa được trình bày trong sách này, ta đã phát triển một chương trình chọn lọc đặc biệt modGA tạo thuận cho ý kiến trên. Hơn nữa, ta có thể đi xa hơn. Mỗi toán tử có độ thích nghi riêng của nó và cũng trải qua một tiến trình tiến hóa nào đó. Nhưng các toán tử được chọn và áp dụng một cách ngẫu nhiên, tùy theo độ thích nghi của chúng. Ý kiến này không phải mới và đã từng xuất hiện, nhưng nó lại mang một ý nghĩa mới và tầm quan trọng mới trong kỹ thuật lập trình tiến hóa.

Một hướng thú vị khác trong việc tìm kiếm cách xử lý lược đồ tốt hơn, đã được nói đến cùng với bài toán lừa, đã được đề nghị gần đây: thuật giải di truyền hỗn loạn (mGA).

mGA khác với GA cổ điển ở nhiều mặt: biểu diễn, các toán tử, kích thước quần thể, chọn lọc, và các giai đoạn của tiến tình tiến hóa. Ta sẽ lần lượt bàn ngắn gọn về chúng.

Trước tiên, mỗi bit nhiễm sắc thể được gắn với tên của nó (số) - cùng một heuristic mà ta đã dùng khi bàn về toán tử đảo. Ngoài ra, các chuỗi có chiều dài thay đổi được và không yêu cầu chuỗi phải có các bổ sung đầy đủ về gen. Chuỗi có thể có các gen lẻ và thừa. Thí dụ, các chuỗi sau đây là các nhiễm sắc thể hợp lệ trong mGA:

v_1 = ((7,1)(1,0))

v_2 = ((3,1)(9,0)(3,1)(3,1)(3,1))

v_3 = ((2,1)(2,0)(4,1)(5,0)(6,0)(7,1)(8,1))

Số đầu trong mỗi cặp ngoặc chỉ vị trí, số thứ hai chỉ giá trị bit. Theo đó, chuỗi thứ nhất v_1 đặc tả hai bit: bit 1 ở vị trí thứ 7 và bit 0 ở vị trí thứ nhất.

Rõ ràng, chiều dài thay đổi được, các chuỗi được đặc tả trên mức hoặc dưới mức đều có thể ảnh hưởng đến các toán tử được sử dụng. Lai đơn giản được thay thế bởi hai toán tử (thậm chí đơn giản hơn): *ghép* và *cắt*. Toán tử ghép nối hai chuỗi đã chọn (bằng các khả năng ghép đặc hiệu). Thí dụ, nối các chuỗi v_1 và v_2 ta được:

v_4 = ((7,1)(1,0)(3,1)(9,0)(3,1)(3,1)(3,1)).

Toán tử cắt (với một xác suất cắt nào đó) chuỗi được chọn tại một vị trí xác định ngẫu nhiên dọc theo chiều dài của nó. Thí dụ, cắt chuỗi v_3 tại vị trí 4 , ta có:

v_5 = ((2,1)(2,0)(4,1)(5,0))

v_6 = ((6,0)(7,1)(8,1))

Hơn nữa, có một toán tử đột biến không đổi, đổi 0 thành 1 (hay ngược lại) với một xác suất xác định nào đó.

Có một số khác biệt khác giữa GA và mGA. Các thuật giải di truyền hỗn loạn (đối với chọn lọc đáng tin cậy bất chấp việc định tỉ lệ hàm) dùng dạng chọn lọc tranh đua; các thuật giải này cũng chia tiến trình tiến hóa thành hai giai đoạn (giai đoạn đầu chọn các khối kiến trúc, và các toán tử di truyền chỉ tham gia trong giai đoạn hai), và kích thước quần thể chỉ thay đổi trong tiến trình.

Phụ lục 2

CHIẾN LƯỢC TIẾN HÓA VÀ CÁC PHƯƠNG PHÁP KHÁC

Các chiến lược tiến hóa (ES) là những thuật giải rập khuôn các nguyên tắc của tiến hóa tự nhiên để giải những bài toán tối ưu hóa có tham số. Chúng được phát triển ở Đức trong thập kỷ 1960.

Các chiến lược tiến hóa ban đầu là những chương trình tiến hóa sử dụng biểu diễn số chấm động, với đột biến là toán tử tái kết hợp duy nhất. Chúng được áp dụng vào những bài toán tối ưu hóa khác nhau với các tham số có thể biến thiên liên tục. Chỉ gần đây, chúng mới được mở rộng cho những bài toán rời rạc.

Trong chương này, ta mô tả ES nguyên thủy dựa trên một quần thể có hai thành viên và toán tử đột biến, và nhiều quần thể có nhiều thành viên ES (phần 1); phần 2 so sánh ES với GA, trong khi phần 4 giới thiệu các chiến lược do nhiều nhà nghiên cứu gần đây đề xuất.

1. TIẾN HÓA CỦA CÁC CHIẾN LƯỢC TIẾN HÓA

Các chiến lược tiến hóa sớm nhất được dựa trên quần thể chỉ có một cá thể. Chỉ có một toán tử di truyền được dùng trong tiến trình tiến hóa. Nhưng, ý tưởng thú vị (không có trong GA) là biểu diễn cá thể bằng một cặp vectơ chấm động, nghĩa là, $v = (x, \sigma)$. Ở đây, vectơ x biểu diễn một điểm trong không gian tìm kiếm; vectơ, σ là vectơ của các độ lệch chuẩn; đột biến được thực hiện bằng các thay x bằng

$$x^{t+1} = x^t + N(0, \sigma)$$

trong đó $N(0, \sigma)$ là vectơ Gauss ngẫu nhiên, độc lập có trung bình là 0 và độ lệch chuẩn σ. (Điều này phù hợp với quan sát sinh học là những thay đổi nhỏ xuất hiện thường xuyên hơn những thay đổi lớn

hơn). Con (cá thể được đột biến) được nhận làm thành viên của quần thể (nó thay cha–mẹ nó) nếu và chỉ nếu nó thích nghi hơn và tất cả các ràng buộc (nếu có) được thỏa. Thí dụ, nếu f là hàm mục tiêu không ràng buộc cần cực đại hóa, một con (x^{t+1}, σ) thay cha–mẹ của nó(x^t, σ) nếu và chỉ nếu $f(x^{t+1}) > f(x^t)$. Nếu không, con sẽ bị loại và quần thể không đổi.

Ta hãy minh họa bước đầu tiên của chiến lược tiến hóa đó bằng cách xét bài toán cực đại hóa đã được dùng làm thí dụ (cho thuật giải di truyền đơn giản) trong chương 2:

$$f(x_1, x_2) = 21.5 + x_1 * \sin(4\pi x_1) + x_2 * \sin(20\pi x_2)$$

với $-3.0 \le x_1 \le 12.1$ và $4.1 \le x_2 \le 5.8$

Như đã giải thích trước, một quần thể chỉ gồm một cá thể duy nhất (x, σ), $x = (x_1, x_2)$ là điểm nằm trong không gian tìm kiếm ($-3.0 \le x_1 \le 12.1$ và $4.1 \le x_2 \le 5.8$) và $\sigma = (\sigma_1, \sigma_2)$ biểu diễn hai độ lệch chuẩn được dùng trong đột biến. Ta giả sử rằng tại một thời điểm t nào đó, quần thể có một phần tử là cá thể sau:

$$(x^t, \sigma) = ((5.3, 4.9), (1.0, 1.0)),$$

và giả sử thêm, đột biến đưa đến thay đổi sau đây:

$$x_1^{t+1} = x_1^t + (0, 1.0) = 5.3 + 0.4 = 5.7$$

$$x_2^{t+1} = x_2^t + (0, 1.0) = 4.9 - 0.3 = 4.6$$

Vì

$$f(x^t) = f(5.3, 4.9) = 18.383705 < 24849532 = f(5.7, 4.6) = f(x^{t+1}),$$

và cả x_1^{t+1} lẫn x_2^{t+1} đều nằm trong các khoảng của chúng, con này sẽ thay cha–mẹ của nó trong quần thể một phần tử. Mặc dù quần thể có một cá thể duy nhất trải qua đột biến, nhưng chiến lược tiến hóa nêu trên lại được gọi là "chiến lược tiến hóa có hai thành viên". Lý

do là con đấu tranh với cha–mẹ và ở giai đoạn tranh đấu có(tạm thời) hai cá thể trong quần thể.

Vectơ của các độ lệch chuẩn σ vẫn không đổi trong tiến trình tiến hóa. Nếu tất cả các thành phần của vectơ này đều giống nhau, nghĩa là $\sigma = (\sigma, ..., \sigma)$, và bài toán tối ưu hóa là *thường lệ*, có thể chứng minh định lý hội tụ:

Định lý 1 (Định lý hội tụ) Với $\sigma > 0$ và một bài toán tối ưu hóa thông thường, có $f_{opt} > -\infty$ (cực tiểu hóa) hoặc $f_{opt} < \infty$ (cực đại hóa)

$$p\{\lim_{t\to\infty} f(x_t) = f_{opt}\} = 1$$

Định lý hội tụ phát biểu rằng tối ưu toàn cục được tìm với xác suất bằng 1, đối với thời gian tìm kiếm đủ dài; nhưng lại không cung cấp những manh mối cho tốc độ hội tụ (thương số của khoảng cách đã vượt được để tiến đến tối ưu và số thế hệ cần thiết đã trải qua để vượt được khoảng cách này). Để tăng tốc độ hội tu, Rechenberg đề nghị một "luật thành công 1/5":

Tỉ lệ φ của đột biến thành công đối với tất cả các đột biến phải là 1/5. Nếu $\varphi > 1/5$, hãy tăng phương sai của toán tử đột biến; ngược lại, hãy giảm nó.

Luật thành công 1/5 xuất hiện như một kết luận của quá trình thực nghiệm tốc độ hội tụ của hai hàm (được gọi là mô hình trải dài và mô hình khối cầu. Luật này được áp dụng mỗi k thế hệ (k là một tham số khác của phương pháp): luật thành công 1/5.

$$\sigma_2^{t+1} = \begin{cases} c_d.\sigma^t, & \text{nếu } \varphi(k) < 1/5 \\ c_i.\sigma^t, & \text{nếu } \varphi(k) > 1/5 \\ \sigma^t, & \text{nếu } \varphi(k) = 1/5 \end{cases}$$

trong đó, $\varphi(k)$ là tỉ lệ thành công của toán tử đột biến trong k thế hệ cuối cùng, và $c_i > 1$ và $c_d < 1$ điều hòa các tốc độ tăng giảm cho phương sai của đột biến. Trong các thử nghiệm của mình, Schwefel đã dùng các giá trị sau: $c_d = 0.82$, $c_i = 1.22 = 1/0.82$.

Lý do trực giác đằng sau luật thành công 1/5 này là hiệu quả tìm kiếm gia tăng: nếu thành công, tìm kiếm sẽ tiếp tục trong các bước "lớn hơn"; nếu không, các bước sẽ ngắn đi, tuy nhiên, tìm kiếm này có thể dẫn đến việc hội tụ sớm đối với một số lớp hàm – điều này đưa đến việc tinh chế phương pháp: kích thước quần thể gia tăng.

Chiến lược tiến hóa có nhiều thành viên khác với chiến lược có hai thành viên trong kích thước của quần thể trước đây (*pop–size* >1). Các tính năng khác của các chiến lược tiến hóa nhiều thành viên là:

- Tất cả các cá thể trong quần thể có cùng xác suất ghép đôi,
- Khả năng đưa vào một toán tử tái kết hợp (trong cộng đồng GA có tên "lai đồng dạng"), ở đây hai cha–mẹ (được chọn ngẫu nhiên):

$(x^1, \sigma^1) = ((x_1^1,\ldots,x_1^n),(\sigma_1^1,\ldots, \sigma_n^1))$ và

$(x^2, \sigma^2) = ((x_1^2,\ldots,x_n^2),(\sigma_1^2,\ldots, \sigma_n^2))$

sinh ra con:

$(x, \sigma) = ((x_1^{q1},\ldots,x_n^{qn}),(\sigma_1^{q1},\ldots, \sigma_n^{qn}))$

trong đó $q_i = 1$ hoặc $q_i = 2$ với cùng xác suất và $i = 1,\ldots, n$.

Toán tử đột biến và hiệu chỉnh của σ vẫn không đổi.

Vẫn có điểm tương đồng giữa chiến lược tiến hóa hai thành viên và nhiều thành viên: cả hai đều sản sinh một con duy nhất. Trong chiến lược hai thành viên, con tranh đấu với cha–mẹ của nó. Trong chiến lược nhiều thành viên, cá thể yếu nhất (trong số *pop–size* +1

cá thể; nghĩa là, các cá thể *pop–size* gốc cộng 1 con) bị loại trừ. Một chú ý có lợi, cũng giải thích được những tinh chế sau này của các chiến lược tiến hóa là:

(1+1) – ES , cho chiến lược tiến hóa hai thành viên, và

$(\mu +1)$ – ES, cho chiến lược tiến hóa nhiều thành viên,

với μ = *pop–size*.

Chiến lược tiến hóa nhiều thành viên phát triển thành:

$(\mu + \lambda)$ – ESs và (μ , λ) – ESs;

Ý chính ẩn sau chiến lược tiến hóa này là cho phép điều chỉnh các tham số (như phương sai đột biến) để tự thích nghi chứ không thay đổi các giá trị của chúng bằng một thuật giải có tính tất định nào đó.

$(\mu + \lambda)$ – ES là mở rộng tự nhiên của chiến lược tiến hóa nhiều thành viên $(\mu +1)$ –ES, ở đây, μ cá thể sản sinh λ con. Quần thể mới (tạm thời) $\mu + \lambda$ cá thể qua quá trình chọn lọc lần nữa còn μ cá thể. Mặt khác, trong (μ , λ) – ES, μ cá thể sản xuất λ con ($\lambda > \mu$) và tiến trình chọn lọc chọn một quần thể mới gồm μ cá thể chỉ từ tập λ con. Làm như vậy, đời sống của mỗi cá thể giới hạn chỉ trong một thế hệ. Điều này cho phép (μ , λ) – ES thực hiện tốt hơn trên những bài toán có tối ưu dịch chuyển hay trên những bài toán mà hàm mục tiêu bị nhiễu.

Các toán tử được dùng trong các $(\mu + \lambda)$ – ES và (μ , λ) – ES kết hợp việc học hai cấp: tham số điều khiển σ của chúng không còn là hằng số, cũng không được thay đổi bằng một thuật giải tất định nào đó (như luật thành công 1/5), nhưng nó được kết hợp vào cấu trúc của các cá thể và trải qua quá trình tiến hóa. Để sản sinh một con, hệ thống hoạt động qua nhiều giai đoạn:

- chọn hai cá thể:

$(x^1, \sigma^1) = ((x_1^1,\ldots,x_1^n),(\sigma_1^1,\ldots, \sigma_n^1))$ và

$(x^2, \sigma^2) = ((x_1^2,...,x_n^2),(\sigma_1^2,..., \sigma_n^2))$

và áp dụng một toán tử (lai tạo, tái kết hợp). Có hai loại lai tạo:

– rời rạc, khi con mới là:

$(x, \sigma) = ((x_1^{q1},...,x_n^{qn}),(\sigma_1^{q1},..., \sigma_n^{qn}))$

ở đây, q_i = 1 hay q_i =2 (vì thế mỗi thành phần xuất phát từ cha–mẹ được chọn trước, thứ nhất hay thứ hai).

– trung gian, khi con mới là:

$(x, \sigma) = ((x_1^1+x_1^2)/2, ..., (x_n^1+x_n^2)/2), ((\sigma_1^1 +\sigma_1^2)/2,..., (\sigma_n^1 +\sigma_n^2)/2))$.

Mỗi toán tử này cũng có thể được áp dụng vào chế độ toàn cục, ở đây cặp cha–mẹ mới được chọn cho mỗi thành phần của vectơ con.

- áp dụng đột biến cho con nhận được (x, σ); con mới nhận được là(x', σ'), ở đây:

$\sigma' = \sigma * e^{N(0,\Delta\sigma)}$ và

$x' = x + N(0,\sigma')$

trong đó $\Delta\sigma$ là tham số của phương pháp.

Để cải thiện tốc độ hội tụ của ES, Schewel đưa vào thêm một tham số điều khiển θ. Điều khiển mới này tạo tương quan giữa các đột biến. Đối với ES đã đề cập từ trước đến giờ (vớiσ_i được chuyên dùng cho mỗi x_i), hướng được ưa thích của tìm kiếm chỉ có thể được thiết lập dọc theo các trục của hệ thống kết hợp. Bây giờ mỗi cá thể trong quần thể được biểu diễn là:

(x,σ,θ).

Các toán tử tái kết hợp cũng giống như những toán tử đã nói đến ở đoạn trên và đột biến tạo con (x', σ', θ') từ (x, σ, θ) theo cách sau:

$\sigma' = \sigma * e^{N(0,\Delta\sigma)}$ và

$\theta' = \theta + N(0, \Delta\theta)$

$x' = x + C(0,\sigma', \theta')$

trong đó $\Delta\theta$ là tham số bổ sung của phương pháp, và $C(0, \sigma', \theta')$ cho thấy vectơ Gauss ngẫu nhiên độc lập với trung bình 0 và mật độ xác suất thích hợp.

Chiến lược tiến hóa rất hiệu quả trong các miền số, vì chúng được (ít nhất là lúc đầu) chuyên dùng cho các bài toán tối ưu hàm (thực). Chúng là các thí dụ về chương trình tiến hóa sử dụng cấu trúc dữ liệu thích hợp (vectơ chấm động) được mở rộng bằng các tham số chiến lược điều khiển) và các toán tử "di truyền" cho bài toán.

Khá thú vị khi so sánh thuật giải di truyền và chiến lược tiến hóa: những khác biệt và tương đồng, những mặt mạnh và mặt yếu của chúng sẽ được bàn đến trong phần sau.

2. SO SÁNH CÁC CHIẾN LƯỢC TIẾN HÓA VÀ CÁC THUẬT GIẢI DI TRUYỀN

Khác biệt cơ bản giữa chiến lược tiến hóa và thuật giải di truyền là ở các lãnh vực áp dụng chúng. Chiến lược tiến hóa được phát triển như những phương pháp tối ưu số. Chúng sử dụng một thủ tục leo đồi đặc biệt với kích thước bước tự thích nghi σ và dốc nghiêng θ . Chỉ gần đây, ES mới được áp dụng cho các bài toán tối ưu rời rạc. Mặt khác, thuật giải di truyền được hình thức hóa thành những kỹ thuật tìm kiếm thích hợp (mục đích chung, cấp phát số lần thử tăng theo lũy thừa cho các lược đồ trên trung bình. GA được áp dụng trong nhiều lãnh vực khác nhau, và một việc tối ưu hóa tham số (thực) chỉ là một lãnh vực của những ứng dụng của chúng).

Vì lý do đó, sẽ không công bằng nếu so sánh thời gian thực hiện và độ chính xác của ES và GA trên cơ sở tối ưu hàm số. Nhưng

cả hai, ES và GA đều là những thí dụ về chương trình tiến hóa, nên nếu bàn một cách tổng quát về những khác biệt và tương đồng của chúng thì cũng hoàn toàn tự nhiên.

Tương đồng chính giữa các ES và GA là cả hai hệ thống đều duy trì các quần thể những lời giải mạnh và tận dụng nguyên tắc chọn lọc tự nhiên. Tuy vậy, giữa hai phương pháp này có rất nhiều khác biệt.

Khác biệt đầu tiên giữa ES và GA cổ điển là cách biểu diễn các cá thể. Như trong nhiều dịp đã được nói đến, ES thao tác trên các vectơ chấm động, trong khi GA cổ điển thao tác trên các vectơ nhị phân.

Khác biệt thứ hai ẩn trong chính tiến trình chọn lọc. Trong một thế hệ của ES, μ cha–mẹ phát sinh quần thể trung gian gồm λ con được sản sinh bằng việc tái kết hợp và các toán tử đột biến (với $(\mu + \lambda)$ – ES), cộng với (đối với (μ , λ) – ES) μ cha–mẹ gốc. Rồi tiến trình chọn lọc giảm kích thước của quần thể trung gian này còn μ cá thể bằng cách loại bỏ những cá thể ít thích nghi nhất trong quần thể. Quần thể gồm μ cá thể này thành lập thế hệ kế tiếp. Trong một thế hệ của GA, thủ tục chọn lọc chọn *pop–size* cá thể từ quần thể được có kích thước *pop–size*. Các cá thể được chọn lặp đi lặp lại, nghĩa là một cá thể mạnh sẽ có cơ hội tốt để được chọn nhiều lần vào quần thể mới. Đồng thời, ngay cả cá thể yếu nhất cũng có cơ hội để được chọn.

Trong ES, thủ tục chọn lọc là tất định: nó chọn μ tốt nhất từ $\mu + \lambda$ ($(\mu + \lambda)$ – ES hoặc (μ , λ) – ES) cá thể (không lặp). Mặt khác, trong GA, thủ tục chọn lọc là ngẫu nhiên, chọn *pop–size* từ *pop–size* cá thể (lặp đi lặp lại), các cơ hội chọn lọc tỉ lệ với độ thích nghi của cá thể. Thực ra một số GA dùng chọn lọc xếp hạng; nhưng các cá thể mạnh vẫn có thể được chọn vài lần. Nói cách khác, việc chọn trong

ES là tĩnh, làm tuyệt chủng và có tính thế hệ, đối với (μ , λ) – ES, trong khi ở GA việc chọn là động, có tính bảo tồn.

Thứ tự tương đối của các thủ tục chọn lọc và tái kết hợp tạo nên khác biệt thứ ba giữa GA và ES: trong ES, tiến trình chọn lọc tiếp theo việc áp dụng các toán tử tái kết hợp, trong khi trong GA các bước này xuất hiện theo thứ tự ngược lại. Trong ES, con là kết quả lai tạo giữa hai cha–mẹ và một đột biến sau đó. Khi quần thể trung gian của $\mu + \lambda$ (hoặc λ) các cá thể đã hoàn tất, thủ tục chọn lọc giảm kích thước của nó còn lại μ cá thể. Trong GA, trước tiên, ta chọn quần thể trung gian. Rồi áp dụng các toán tử di truyền (lai tạo và đột biến) cho một số cá thể (được chọn theo xác suất lai tạo) và một số gen (chọn theo xác suất đột biến).

Khác biệt kế tiếp giữa ES và GA là các tham số sinh sản của GA (xác suất lai tạo, xác suất đột biến) giữ nguyên không đổi trong suốt quá trình tiến hóa, trong khi ES thay đổi chúng (σ và θ) luôn: chúng trải qua đột biến và lai tạo cùng với vectơ lời giải x, do một cá thể được hiểu là một bộ ba (x, σ, θ). Điều này rất quan trọng – tính tự thích nghi của các tham số điều khiển trong ES chịu trách nhiệm dò tìm chính xác cục bộ của hệ thống.

ES và GA cũng xử lý các ràng buộc theo cách khác nhau. Chiến lược tiến hóa nhận một tập các bất đẳng thức $q \geq 0$:

$$g_1(x) \geq 0, \ldots, g_q(x) \geq 0,$$

là một phần của bài toán tối ưu hóa. Nếu trong lần lặp nào đó, có một con không thỏa tất cả các ràng buộc này, thì con đó không đủ phẩm chất, nghĩa là nó không được chọn vào quần thể mới. Nếu tỉ lệ xuất hiện những con bất hợp lệ như thế cao thì ES điều chỉnh những tham số điều khiển của chúng, như bằng cách giảm các thành phần của vectơ σ. Chiến lược chính của thuật giải di truyền để xử lý các ràng buộc là phạt những cá thể vi phạm. Lý do là vì đối với những

bài toán có ràng buộc phức tạp, ta không thể bỏ qua các con bất hợp lệ (GA không điều chỉnh các tham số điều khiển của chúng – ngược lại thuật giải sẽ không tiến hóa trong phần lớn thời gian). Đồng thời, thường các bộ giải mã hay các thuật giải sửa chữa có chi phí quá cao, không thể xét đến (những nỗ lực để xây dựng một thuật giải sửa chữa tốt cũng bằng nỗ lực để giải chính bài toán). Kỹ thuật hàm thưởng phạt có nhiều bất lợi, một trong những bất lợi đó là sự phụ thuộc vào bài toán.

Những điều vừa nói trên đây bao hàm rằng ES và GA hoàn toàn khác biệt theo nhiều chi tiết. Như, khi xem phát triển của ES và GA kỹ hơn trong 20 năm sau này, ta phải công nhận rằng khoảng cách giữa những phương pháp này ngày càng thu nhỏ lại.

Một lần nữa, ta lại bàn về những vấn đề chung quanh các ES và GA, nhưng lần này từ bối cảnh lịch sử.

Ban đầu, có những dấu hiệu cho thấy thuật giải di truyền tỏ ra khó thực hiện việc tìm kiếm đối với những ứng dụng số. Nhiều nhà nghiên cứu đã thử nghiệm với những biểu diễn khác nhau (mã Gray, số chấm động và những toán tử khác nhau để cải thiện hiệu quả của hệ thống dựa trên GA. Ngày nay, khác biệt đầu tiên giữa GA và ES không còn là những vấn đề này nữa: hầu hết các ứng dụng GA cho những bài toán tối ưu hóa sử dụng biểu diễn chấm động, áp dụng các toán tử theo cách thích hợp. Dường như cộng đồng GA vay mượn ý tưởng về biểu diễn vectơ từ ES.

Những kết quả từ những thử nghiệm của chúng ta với các chương trình tiến hóa cung cấp một nhận xét thú vị: chỉ lai tạo hoặc chỉ đột biến thì không thỏa tiến trình tiến hóa. Cả hai toán tử (hoặc cả hai họ các toán tử này) đều cần thiết cho hiệu quả của hệ thống. Các toán tử lai tạo rất quan trọng trong việc khảo sát những vùng hứa hẹn trong không gian tìm kiếm và chịu trách nhiệm về hội tụ sớm hơn (nhưng không quá sớm); trong nhiều hệ thống – đặc biệt

đối với những người làm việc trên những cấu trúc dữ liệu phong phú hơn, việc giảm tỉ lệ lai tạo làm hiệu quả của chúng kém đi. Đồng thời, xác suất áp dụng các toán tử đột biến lại thật cao: hệ thống GENETIC –2 sử dụng tỉ lệ đột biến cao là 0.2.

Cộng đồng ES cũng đạt đến kết luận tương tự. Do đó, toán tử lai tạo được đưa vào ES. Chú ý rằng ES ban đầu chỉ dựa trên một toán tử đột biến và toán tử lai tạo chỉ được kết hợp mãi sau này. Dường như tỉ số giữa GA và ES bằng nhau: cộng đồng ES vay mượn ý tưởng về toán tử lai tạo từ GA.

Có thêm nhiều vấn đề thú vị về mối liên quan giữa ES và GA. Gần đây, một số toán tử lai tạo khác đồng thời được đưa vào GA và ES. Hai vectơ x_1 và x_2 có thể sản sinh hai con y_1 và y_2, là tổ hợp tuyến tính của các cha–mẹ của chúng, nghĩa là:

$y_1 = ax_1 + (1-a)x_2$ và

$y_2 = (1-a)x_1 + ax_2$.

Lai tạo như thế được gọi là:

- trong GA: *lai trung bình bảo đảm* ràng buộc (khi $a = 1/2$), hay *lai số học*, và
- trong ES: *lai tạo trung gian*.

Việc tự thích nghi của các tham số điều khiển trong ES cũng có phần tương ứng của nó trong nghiên cứu GA, ý nghĩ áp dụng thuật giải di truyền khi đang chạy trong ES đã được phát biểu trước đây; hệ thống Argot áp dụng biểu diễn của những cá thể. Bài toán áp dụng các tham số điều khiển cho thuật giải di truyền cũng được nhận thức vào lúc đó. Rõ ràng là việc tìm các thiết lập tốt cho các tham số GA đối với một bài toán đặc biệt không phải là một tác vụ tầm thường. Nhiều phương pháp đã được đề nghị. Một số phương pháp sử dụng thuật giải di truyền giám sát để tối ưu hóa các tham số của thuật giải di truyền "thích đáng" đối với một lớp bài toán. Các

tham số được xét là kích thước quần thể, tỉ lệ lai tạo, tỉ lệ đột biến, cách biệt thế hệ (phần trăm quần thể bị thay thế trong mỗi thế hệ), cửa sổ định tỉ lệ, và một chiến lược chọn lọc (tinh khiết hoặc ưu tú). Phương pháp khác bao gồm việc áp dụng các xác suất của các toán tử di truyền: từ ý tưởng là xác suất áp dụng một toán tử được thay đổi theo tỉ lệ để hiệu quả của các cá thể được tăng theo toán tử này. Trực giác cho rằng các toán tử hiện đang "thực hiện tốt" nên được sử dụng thường xuyên hơn. Nhiều tác giả thử nghiệm với 4 chiến lược khác nhau để cấp phát xác suất của toán tử đột biến: (1) xác suất không đổi, (2) giảm theo lũy thừa, (3) tăng theo lũy thừa và (4) kết hợp (2) và (3).

Cũng vậy, nếu nhớ lại đột biến không đồng dạng, ta sẽ nhận thấy rằng toán tử thay đổi hành động của nó trong quá trình tiến hóa.

Ta hãy so sánh chương trình tiến hóa dựa trên di truyền GENOCOP với một chiến lược tiến hóa một cách ngắn gọn. Cả hai hệ thống duy trì các quần thể lời giải mạnh và dùng thủ tục chọn lọc để phân biệt giữa các cá thể 'tốt' và 'xấu'. Cả hai hệ thống đều dùng biểu diễn số chấm động. Chúng cho độ chính xác cao (qua việc áp dụng các tham số điều khiển – đối với ES; và qua đột biến không đồng dạng – đối với GENOCOP). Cả hai hệ thống xử lý các ràng buộc một cách tinh tế: GENOCOP tận dụng sự hiện diện của các ràng buộc tuyến tính, ES thực hiện trên tập các bất đẳng thức. Cả hai hệ thống đều kết hợp 'ý tưởng xử lý ràng buộc' của nhau. Các toán tử cũng tương tự. Một hệ thống dùng lai tạo trung gian, hệ thống kia dùng lai tạo số học. Chúng có thực sự khác nhau không ?

Nhiều tác giả đã đưa ra một so sánh thú vị giữa ES và GA từ viễn cảnh của các chiến lược tiến hóa.

Một vài năm trước, một kỹ thuật qui hoạch tiến hóa (EP) đã được tổng quát hóa để xử lý các bài toán tối ưu số. Chúng hoàn toàn

tương tự các lan truyền ngược tiến hóa; chúng dùng biểu diễn chấm động và đột biến làm toán tử mấu chốt. Khác biệt cơ bản giữa chiến lược tiến hóa và các kỹ thuật tiến hóa có thể tóm tắt như sau:

- EP không dùng các toán tử tái kết hợp,
- EP sử dụng chọn lọc theo xác suất (chọn lọc đấu tranh) trong khi ES chọn μ cá thể tốt nhất cho thế hệ kế tiếp,
- trong EP các giá trị thích nghi nhận được từ các giá trị hàm mục tiêu bằng cách định tỉ lệ chúng và có lẽ bằng cách tạo một số thay đổi ngẫu nhiên,
- độ lệch chuẩn cho đột biến của mỗi cá thể được tính là căn bậc hai của một biến đổi tuyến tính của giá trị thích nghi của chính nó.

3. TỐI ƯU HÓA HÀM ĐA MỤC TIÊU VÀ ĐA KẾT QUẢ

Trong hầu hết các chương, ta giới thiệu các phương pháp tối ưu đơn, toàn cục của một hàm. Nhưng trong nhiều trường hợp, một hàm có thể có nhiều tối ưu cần được xác định (tối ưu hóa đa kết quả) hoặc có nhiều hơn một chuẩn để tối ưu (tối ưu đa mục tiêu). Rõ ràng, rất cần những kỹ thuật mới để tiếp cận những loại bài toán này; ta sẽ lần lượt bàn về chúng.

3.1. Tối ưu hóa đa kết quả

Trong nhiều ứng dụng, việc định vị tất cả lời giải tối ưu của một hàm cho trước có thể rất quan trọng. Một số phương pháp dựa trên các kỹ thuật tiến hóa đã được đề nghị cho phương pháp tối ưu đa kết quả này.

Kỹ thuật thứ nhất dựa trên việc lặp: ta chỉ việc chạy nhiều lần thuật giải. Nếu tất cả tối ưu có cùng khả năng được tìm ra như nhau, số lần chạy độc lập sẽ là:

$$p\sum_{i-1}^{p}\frac{1}{i} \approx p(\gamma + \log p),$$

trong đó p là số tối ưu và $\gamma \approx 0.577$ là hằng số Euler. Không may, trong hầu hết các ứng dụng thực tế, các tối ưu lại không có khả năng như nhau, vì vậy số lần chạy độc lập phải cao hơn. Cũng có thể dùng một cài đặt song song phương pháp lặp này, lúc đó nhiều quần thể con sẽ tiến hóa cùng lúc (theo cách độc lập, nghĩa là không liên lạc nhau).

Golberg và Richardson mô tả một phương pháp dựa trên sự dùng chung; phương pháp này cho phép việc thành lập các quần thể con ổn định (các chủng loại) của nhiều chuỗi khác nhau – bằng cách này thuật giải khám phá được nhiều đỉnh cùng lúc. Hàm dùng chung xác định việc giảm mức thích nghi của một cá thể do lân cận của nó ở khoảng cách "*dist*" nào đó. Hàm dùng chung *sh* được định nghĩa là hàm theo khoảng cách có những thuộc tính sau:

- $0 \le sh(dist) \le 1$, với mọi quãng *dist*
- $sh(0) = 1$ và
- $\lim_{dist \to \infty} sh(dist) = 0$

Có nhiều hàm dùng chung thỏa các điều kiện trên, một khả năng là:

$$sh(dist) = \begin{cases} 1 - \left(\dfrac{dist}{\sigma_{sh}}\right)^{\alpha}, & \text{nếu dist} < \sigma_{sh} \\ 0, & \text{ngược lại} \end{cases}$$

trong đó σ_{sh} và α là những hằng số.

Độ thích nghi (dùng chung) mới của một cá thể x được cho bởi:

$$eval'(x) = eval(x) / m(x)$$

trong đó *m(x)* trả về số đếm vừa ý cho một cá thể x cụ thể:

$$m(x) = \sum_{y} sh(dist(x, y))$$

Trong công thức trên, tổng tất cả các y trong quần thể bao gồm chính chuỗi x; do đó, nếu chuỗi x bản thân nó đã tự vừa, giá trị thích nghi của nó không giảm *(m(x) =1)*. Ngược lại, hàm thích nghi sẽ giảm theo lũy thừa với số và độ gần với các lân cận.

Nghĩa là, khi nhiều cá thể ở gần nhau, chúng đóng góp vào số đếm chung của nhau, như vậy làm giảm các giá trị thích nghi của nhau. Do đó, kỹ thuật này giới hạn sự gia tăng vô tổ chức của các chủng loại cụ thể trong quần thể.

Gần đây, Beasly, Bull và Martin đã mô tả một kỹ thuật mới (được gọi là sự vừa vặn liên tiếp) cho tối ưu đa kết quả hầu tránh những bất lợi của phương pháp dùng chung này (như, độ phức tạp về thời gian do những tính toán dùng chung thích nghi, kích thước quần thể, tỉ lệ với số tối ưu). Thuật giải được đề nghị này cũng dùng hàm khoảng cách *dist* và hàm thích nghi *eval* và nó dựa trên những ý tưởng sau đây: khi tìm được tối ưu, hàm lượng giá có thể được hiệu chỉnh để loại bỏ lời giải (đã tìm được) này, vì tìm lại cùng một tối ưu chẳng thú vị chút nào. Ở một nghĩa nào đó, những lần chạy kế tiếp của thuật giải di truyền có sử dụng tri thức có được trong những lần chạy trước (ngược lại với kỹ thuật lặp đơn giản: mỗi lần chạy khởi đầu bằng một quần thể được phát sinh ngẫu nhiên). Các bước cơ bản của thuật giải này là:

1. Khởi tạo: cân bằng hàm thích nghi đã biến đổi bằng hàm thích nghi thô.
2. Chạy GA với hàm thích nghi đã biến đổi, giữ thông tin về cá thể tốt nhất tìm được trong lần chạy này.

3. Cập nhật hàm thích nghi đã biến đổi để tạo sự suy thoái trong vùng gần cá thể tốt nhất, tạo ra một hàm thích nghi mới.
4. Nếu độ thích nghi thô của cá thể tốt nhất được ưa thích, (nghĩa là nó vượt ngưỡng lời giải), hãy đưa nó ra làm lời giải.
5. Nếu không tìm được tất cả lời giải, trở về bước hai.

Nhưng gần đây Spears lại đề nghị một phương pháp khác. Thuật giải được đề nghị này thực hiện những ý tưởng về dùng chung và về giao phối hạn chế. Nhưng loại bỏ ý niệm về khoảng cách mêtric mà thay bằng khái niệm về nhãn: mỗi cá thể trong quần thể có một nhãn (trong các thử nghiệm đã được báo cáo, nhãn là một chuỗi n–bit, do đó, các nhãn có thể được dùng để biểu diễn 2^n quần thể con):

Giả sử một hàm đơn giản có hai đỉnh, một đỉnh cao gấp hai đỉnh kia và giả sử thêm rằng ta cho phép một bit gắn thêm vào mỗi cá thể. Mỗi bit gắn thêm này được khởi tạo ngẫu nhiên, vì thế lúc bắt đầu chạy ta có hai quần thể con gần như cùng kích thước. Do được tạo mẫu ngẫu nhiên, cả hai quần thể con cuối cùng cũng ổn định trên đỉnh cao hơn, hay cả hai cùng cư trú trên đỉnh thấp hơn. Nhưng trong một số trường hợp (vẫn do tạo mẫu ngẫu nhiên), mỗi quần thể con sẽ hướng về hai đỉnh khác nhau, nếu ta không có sự dùng chung thích nghi, các cá thể trên đỉnh cao hơn luôn luôn có nhiều con hơn các cá thể trên đỉnh thấp hơn và cuối cùng quần thể con trên đỉnh thấp sẽ biến mất. Nhưng nếu có dùng chung thích nghi, đỉnh cao hơn chỉ có thể mang được số cá thể nhiều hơn đỉnh thấp hai lần (vì nó chỉ cao gấp hai lần). Cơ chế dùng chung thích nghi đã điều chỉnh động tính thích nghi nhận được sao cho hai đỉnh coi như có cùng độ cao. Kết quả là cả hai quần thể con đều sinh tồn một cách vững chắc. Hơn nữa, sự giao phối hạn chế ngăn ngừa việc lai tạo giữa những cá thể trên hai đỉnh, điều này thường đưa đến những cá thể có thích nghi thấp.

3.2. Tối ưu đa mục tiêu

Đối với nhiều bài toán ra quyết định trong thế giới thực, có một nhu cầu là tối ưu hóa đồng thời nhiều mục tiêu.

Những bài toán tối ưu đa mục tiêu này cần những kỹ thuật riêng biệt, mà những kỹ thuật này rất khác với các kỹ thuật tối ưu hóa chuẩn, đối với tối ưu hóa một mục tiêu duy nhất. Rõ ràng là nếu có hai mục tiêu cần tối ưu hóa, ta có thể tìm một lời giải tốt nhất tương xứng với mục tiêu thứ nhất, còn lời giải kia là tốt nhất đối với mục tiêu thứ hai.

Sẽ tiện hơn nếu phân loại tất cả những lời giải mạnh cho bài toán tối ưu hóa đa mục tiêu thành những lời giải bị thống trị và không bị thống trị (hay Pareto–optimal). Vì lời giải x sẽ bị thống trị nếu tại đó tồn tại lời giải khả thi y không kém x trên mọi tọa độ, nghĩa là, đối với mọi hàm mục tiêu f_i ($i = 1, \ldots, k$):

$f_i(x) \le f_i(y)$ với mọi $1 \le i \le k$

Nếu lời giải không bị thống trị bởi bất cứ lời giải khả thi nào khác, ta gọi nó là lời giải không bị thống trị (hoặc Pareto–optimal). Tất cả các lời giải Pareto–optimal có lẽ cho ta một vài lợi ích; lý tưởng thì hệ thống có thể báo cáo lại tập của mọi điểm Pareto–optimal.

Có một số phương pháp cổ điển về tối ưu đa mục tiêu. Những phương pháp này bao gồm phương pháp trọng số hóa các mục tiêu, ở đó các hàm mục tiêu f_i được tổ hợp thành một hàm mục tiêu chung F:

$$F(x) = \sum_{i=1}^{k} w_i f_i(x)$$

trong đó các trọng $w_i \in [0..1]$ và $\sum_{i=1}^{k} w_i = 1$. Các vectơ trọng khác nhau cho các lời giải Pareto – optimal khác nhau. Một phương pháp khác (phương pháp hàm khoảng cách) tổ hợp nhiều hàm mục tiêu vào một hàm trên cơ sở của cấp cần có của vectơ $\boldsymbol{y}$:

$$F(x) = \left(\sum_{i=1}^{k} \left| f_i(x) - y_i \right|^r \right)^{\frac{1}{r}}$$

(thường) $r = 2$ (không gian mêtric Euclide).

Tối ưu đa mục tiêu cũng hưởng lợi từ cộng đồng GA. Vào 1984, Schaffer phát triển chương trình VEGA (Vector Evaluated Genetic Algorithm: thuật giải di truyền vectơ lượng giá), là mở rộng của chương trình GENESIS bao gồm các hàm đa mục tiêu. Ý chính hỗ trợ chương trình VEGA là chia quần thể thành các quần thể con (cùng kích thước); mỗi quần thể con "chịu trách nhiệm" về một mục tiêu duy nhất. Thủ tục chọn lọc được thực hiện độc lập cho mỗi mục tiêu, nhưng lai tạo lại được thực thi ngang qua các biên của quần thể con. Những heuristic bổ sung cũng được phát triển như phương án phân phối lại tài sản, kế hoạch lai giống và nghiên cứu cách giảm khuynh hướng hội tụ của hệ thống về những cá thể không là tốt nhất đối với mỗi mục tiêu.

Srinivas và Deb đề nghị một kỹ thuật mới NSGA (Non–dominated Sorting GA: thuật giải di truyền sắp xếp không thống trị) dựa trên nhiều tầng phân loại các cá thể. Trước khi hệ thống hoạt động, quần thể được xếp hạng trên cơ sở không bị thống trị: tất cả những cá thể không bị thống trị được phân vào một loại (với giá trị thích nghi giả, tỉ lệ với kích thước quần thể, để cung cấp một tiềm năng sinh sản bằng nhau cho những cá thể này). Để duy trì tính đa dạng của quần thể, những cá thể được phân loại được chia xẻ những giá trị thích nghi giả của chúng (xem phần trước). Rồi nhóm các cá

thể đã phân loại này không được quan tâm nữa để xét đến một lớp các cá thể không bị thống trị khác. Tiến trình tiếp tục cho đến tất cả cá thể trong quần thể được phân loại.

Gần đây, Fonseca và Fleming công bố một nghiên cứu về các thuật giải tiến hóa về tối ưu đa mục tiêu. Họ cung cấp một tổng quan về cả hai loại kỹ thuật (những kỹ thuật này kết hợp nhiều tiêu chuẩn vào một hàm mục tiêu và trả về một giá trị duy nhất, và những kỹ thuật dựa trên Pareto–optimal lại trả về một tập các giá trị), cũng như những vấn đề nghiên cứu mở.

4. NHỮNG CHƯƠNG TRÌNH TIẾN HÓA KHÁC

Như đã nói từ trước, những thuật giải di truyền (cổ điển) không thích hợp để dò tìm lời giải chính xác cao, vì thế đối với những bài toán tối ưu số chẳng hạn, GA cho ra những lời giải kém chính xác hơn ES, trừ phi biểu diễn những cá thể trong GA được đổi từ nhị phân thành chấm động và hệ thống (tiến hóa) cung cấp các toán tử chuyên biệt (như đột biến không đồng dạng. Nhưng, trong thập kỷ cuối, đã có một số nỗ lực để cải thiện (trực tiếp hoặc gián tiếp) đặc trưng này của GA.

Một hiệu chỉnh thú vị của GA, là mã hóa Delta, do Whitley đề nghị gần đây. Ý tưởng chính của chiến lược này là nó coi những cá thể trong quần thể không là các lời giải mạnh của bài toán, mà là những giá trị (nhỏ) bổ sung (gọi là *các giá trị delta*), để bổ sung vào lời giải mạnh hiện hành. Thuật giải mã hóa Delta (được giản lược hóa) được mô tả trong hình 1.

---

thủ tục Delta Coding

bắt đầu

áp dụng GA ở cấp x

lưu lời giải tốt nhất (x)

khi (điều kiện dừng chưa thỏa) **làm**

bắt đầu

áp dụng GA ở cấp δ

lưu lời giải tốt nhất (δ)

biến đổi lời giải tốt nhất (cấp x):

$x \leftarrow x + \delta$

hết lặp

kết thúc

Hình P.1. Một thuật giải mã hóa Delta (đã giản lược).

Thuật giải mã hóa Delta ứng dụng những kỹ thuật thuật giải di truyền theo hai mức: mức các lời giải mạnh của bài toán (mức x) và (giai đoạn lặp) mức các thay đổi delta (mức δ). Lời giải mạnh nhất tìm được ở mức x bằng cách dụng GA được lưu (x) và được giữ làm điểm tham chiếu. Rồi nhiều lần lặp của GA bên trong được thực thi (mức δ). Kết thúc của một lần thực hiện của GA theo mức này (nghĩa là khi GA hội tụ) cho ra vectơ hiệu chỉnh tốt nhất δ, cập nhật các giá trị của x. Sau khi cập nhật, lần lặp kế tiếp diễn ra. Mỗi lần áp dụng GA trong giai đoạn lặp lại khởi tạo quần thể của các δ một cách ngẫu nhiên. Đương nhiên ta lượng giá $x+\delta$ để lượng giá cá thể δ.

Thuật giải mã hóa Delta gốc phức tạp hơn, vì nó thao tác trên các chuỗi bit. Làm như vậy, mã hóa Delta bảo tồn được những nền tảng lý thuyết của thuật giải di truyền (vì tại mỗi lần lặp lại có một lần chạy GA). Các điều kiện dừng của các GA ở cả hai mức được biểu diễn theo khoảng cách Hamming giữa phần tử tốt nhất và kém nhất trong quần thể (thuật giải dừng khi khoảng cách Hamming không lớn hơn 1). Ngoài ra, có một biến *len* để giải mã số bit biểu diễn một thành phần của vectơ δ (thực ra, chỉ có *len* -1 là biểu diễn giá trị tuyệt đối của thành phần; bit cuối cùng là bit dấu). Nếu lời giải tốt nhất từ cấp δ đạt được vectơ

$\delta = (0,0,\ldots,0)$,

(nghĩa là, không có thay đổi ở lời giải mạnh tốt nhất), biến *len* được tăng lên 1 (để tăng độ chính xác của lời giải), ngược lại nó sẽ bị giảm 1. Cũng ghi nhận rằng mã δ khiến cho các đột biến trở thành không cần thiết, do những lần khởi tạo lại của các quần thể theo mức δ cho mỗi lần lặp.

Ta có thể giản lược thuật giải mã hóa Delta gốc (hình 1 đã cho một thí dụ giản lược như vậy) và cải thiện độ chính xác cũng như thời gian thực hiện của nó, nếu ta biểu diễn cả hai vectơ x và δ bằng các chuỗi số chấm động.

Ý kiến về việc khởi tạo lại quần thể đã được bàn đến Golbergs khám phá các thuộc tính của hệ thống: dùng kích thước quần thể nhỏ nhưng khởi tạo lại nó mỗi khi thuật giải di truyền hội tụ (và dĩ nhiên lưu những cá thể tốt nhất!). Xem hình 2 để biết được về chiến lược này (gọi là chọn theo chuỗi).

thủ tục Chọn hàng loạt

bắt đầu

phát sinh một quần thể (nhỏ)

khi (điều kiện dừng chưa thỏa) **làm**

bắt đầu

áp dụng GA

lưu lời giải tốt nhất *(x)*

phát sinh một quần thể mới bằng cách truyền

các cá thể tốt nhất của quần thể được

hội tụ rồi phát sinh các cá thể còn lại

một cách ngẫu nhiên.

hết lặp

kết thúc

Hình P.2. GA dựa trên quần thể được khởi tạo lại.

Việc khởi tạo lại các quần thể làm cho các cá thể có tính đa dạng và giúp hệ thống có hiệu quả tích cực.

Một số chiến lược được đề nghị có bao gồm một thành phần học, cũng theo cùng cách như trong các chiến lược tiến hóa. Grefenstette đề nghị các tham số điều khiển tối ưu hóa cho thuật giải di truyền (kích thước quần thể, các tỉ lệ lai tạo và đột biến, vv...) bằng một thuật giải khác, thuật giải di truyền giám sát. Shaefer bàn về chiến

lược ARGOT (Adaptive Representation Genetic Optimizer Technique), mà hệ thống học cách biểu diễn bên trong tốt nhất cho các cá thể.

Một chương trình tiến hóa thú vị khác là chiến lược tiến hóa chọn lọc (IRM, Immune Recruitment Mechanism: Cơ chế tuyển chọn miễn dịch), được đề nghị gần đây làm kỹ thuật tối ưu hóa trong những không gian thực (một hệ thống tương tự cho các hàm tối ưu hóa trong các không gian Hamming là GIRM). Chiến lược kết hợp một số ý tưởng đã có trước đây để hướng điều khiển theo hướng ta muốn. Như trong tất cả kỹ thuật lập trình tiến hóa, con được phát sinh từ quần thể hiện hành. trong những thuật giải di truyền cổ điển, con thay chỗ cha–mẹ. Trong chiến lược tiến hóa, con đấu tranh với cha–mẹ (các ES đầu tiên), nó đấu tranh với cha–mẹ và những con khác ($(\mu+\lambda)$ –ES), hay tranh đấu với các con khác ((μ,λ) –ES). Trong hệ thống IRM, con phải qua một thử nghiệm bổ sung về mối quan hệ chặt chẽ với các láng giềng. Kiểm tra muốn biết nó có đủ tương đồng với những láng giềng gần của nó không.

Nói chung, một ứng viên *k* chỉ có thể qua được kiểm tra quan hệ này, nếu:

$$\sum_i m(k,i).f_i > T$$

trong đó, *i* biểu thị các chủng loại khác đã có trong quần thể, f_i là tập trung của các chủng loại *i*, còn *m(k,i)* là hàm quan hệ của các chủng loại *k* và *i*, cuối cùng, *T* là ngưỡng tuyển chọn.

Chiến lược IRM chỉ đạo tìm kiếm bằng cách chỉ nhận những cá thể thỏa kiểm tra quan hệ. Glover trong Scatter and Tabu Search đã hình thức hóa những ý tưởng tương tự. Những kỹ thuật tìm kiếm Scatter, giống các chương trình tiến hóa khác, duy trì một quần thể các lời giải mạnh (vectơ x^i là các điểm tham chiếu). Chiến lược này thống nhất các tập con được ưa thích của các điểm tham chiếu để phát sinh những điểm thử nghiệm (con) bằng những tổ hợp tuyến

tính có trọng, và chọn những phần tử tốt nhất làm nguồn của các điểm tham chiếu mới (quần thể mới). Một khuynh hướng mới ở đây là việc dùng đa lai tạo (được gọi là tổ hợp trọng), mà nhiều (hơn 2) cha–mẹ góp sức để tạo một con. Glover mở rộng ý tưởng về tìm kiếm Scatter bằng cách kết hợp nó với tìm kiếm Tabu – kỹ thuật này hạn chế việc chọn con mới (nó đòi hỏi bộ nhớ nơi giữ tập cá thể tiền sử. Hình 3 trình bày cấu trúc của thuật giải tìm kiếm scatter/tabu.

Sau khi khởi tạo và lượng giá thuật giải tìm kiếm scatter/tabu phân loại (phân loại bước *P(t)*) quần thể các lời giải $\bar{X}_1, \ldots, \bar{X}_{pop_size}$ thành nhiều tập. Gồm có (1) tập các phần tử phát sinh tiền sử tối ưu *V* chứa một số (cố định) các lời giải tốt nhất qua toàn bộ quá trình, (2) tập các phần tử phát sinh tabu $T \subseteq V$ chứa những lời giải hiện không được xét đến, (3) tập các phần tử phát sinh tiền sử được chọn V^* chứa những phần tử tốt nhất của $V - T$, và (4) tập các phần tử phát sinh đang được chọn S^* chứa những phần tử tốt nhất của *S*. Bước phân loại (phân loại *P(t)*) sau này được lặp lại trong giai đoạn lặp của thuật giải.

Trong mỗi lần lặp, một tập *R(t)* các điểm thử nghiệm lại được tạo. Các điểm thử nghiệm tương ứng với số con của quần thể *P(t)*; chúng được lượng giá và (một số) được kết hợp vào quần thể mới (chọn P(t) từ *P(t–1)* và *R(t)*).

---

thủ tục tìm kiếm scatter/ tabu

bắt đầu

t = 0

khởi tạo *P(t)*

lượng giá *P(t)*

khi (điều kiện dừng chưa thỏa) **làm**

bắt đầu

t = t+1

tạo ra *R(t)*

lượng giá *R(t)*

chọn *P(t)* từ *P(t–1)* và *R(t)*

phân loại* *P(t)*

hết lặp

kết thúc

Hình P.3. tìm kiếm scatter/ tabu

Gần đây, David Fogel áp dụng những ý tưởng về lập trình tiến hóa vào tối ưu hóa liên tục giá trị thực mở rộng này gộp vào các biến độc lập tự thích nghi và những thủ tục tối ưu hóa ma trận đồng phương sai.

Maniezzo phát triển khái niệm tiến hóa *nổi hạt*, mà thuật giải cho phép tiến hóa các mẫu hàm mục tiêu đồng thời với phân giải sự tạo mẫu (nghĩa là, sự nổi hạt). Chiều dài của các cá thể trở nên thay đổi được mà việc mã hóa được thông dịch theo cấp phân giải cụ thể được đặc tả trong nhiễm sắc thể.

Một khái niệm của thuật giải di truyền xử lý quãng (thuật giải di truyền quãng Interval Genetic Algorithm) cũng được Muselli và Ridella khám phá. Thuật giải di truyền quãng kết hợp ý tưởng của những thuật giải di truyền và luyện thép mô phỏng; các toán tử di truyền (lai tạo, phát sinh một quãng mới từ hai quãng; sát nhập,

phát sinh con từ hai cha–mẹ là chỗ giao nhau của hai quãng và đột biến, tìm kiếm quãng (nghĩa là một cha hoặc mẹ) cho điểm tốt nhất.

Dường như, hướng tìm kiếm nhiều hứa hẹn nhất của " bộ tối ưu hóa tối ưu" nằm ở đâu đó trong những ý tương trên. Mỗi chiến lược cung cấp một hiểu biết mới, có thể hữu ích cho việc phát triển một chương trình tiến hóa đối với một số lớp bài toán.

TÀI LIỆU THAM KHẢO

[Bak96] Thomas Back. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996.

[Deb99] K. Deb. *Evolutionary Multi-Critierion Optimization*. John Wiley and Sons, 1999.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addition Wesley, 1989

[MDS99] Z. Michalewicz, K. Deb, M. Schmidt and Th. Stidsen. *Evolutionary Algorithms for Engineering Applications*. John Wiley and Sons, 1999.

[Mic99] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1999.

[SD 99] Th. Stutzle and M. Dorigo. *ACO Algorithms for the Traveling Salesman Problem*. John Wiley and Sons, 1999.

ÂN TẠO

tiến hóa

thuật giải di truyền**. Lý thuyết, nguyên lý và cơ ch
i di truyền được phân tích và giải thích tường tận và d
u cụ thể.

uyền giải **bài toán tối ưu hàm số với các ràng buộ
c kỹ thuật xử lý ràng buộc được xem xét và phân tíc

n tối ưu tổ hợp phổ biến đã được xếp vào lớp các bà
c xem xét và giải quyết bằng kỹ thuật lập trình tiến hóa





Giá : 29.000 đ